Deploying Apereo CAS

Last generated: October 18, 2018

Table of Contents

Adding SAML support

Building the SAML client

Enabling the dashboard (adminpages)

Building the management webapp

Introduction

Summary: This document provides step-by-step instructions for setting up an Apereo CAS 5 environment. It was created during the process of building a brand new development environment to experiment with many of the new features in this release.

CAS VERSION USED IN THIS DOCUMENT

The instructions, configuration settings, and hyperlink references contained in this document are based on **CAS 5.2.7**. Most of the information presented here is also applicable to the CAS 5.1.*x* and CAS 5.3.*x* branches, with the caveat that some features may not exist in earlier versions, and the names or values of some configuration settings may have changed between releases.

The best way to learn about changes from one release to another is to read Misagh Moayyed's "Changelog" blog posts. These come out with each release candidate, and describe the significant changes from the previous release candidate (cumulatively, the changelogs for the several release candidates describe the changes from one feature release to the next). To access these blog posts, go to the Releases section of the CAS GitHub Repository and search for "Changelog".

Warning: The instructions here *will not* work for building, configuring, or deploying CAS 6.

Apereo CAS 5 was released in November 2016. The new release improved on many of the enhancements introduced in the CAS 4 series of releases, and also introduced several new features that will enable The New School to offer an improved single sign-on experience to its users.

This document provides step-by-step instructions for setting up Apereo CAS 5. It was created to record the configuration choices made, and deployment lessons learned, during the process of building a brand new development environment to experiment with many of CAS 5's new features.

The New School's major implementation goals for this environment are:

- Apply lessons learned from our CAS 3.5 environment

    - "If we had it to do over again, we'd do this differently." This is our chance.

- High availability (fault-tolerant) everything
    - Main CAS servers reside behind load balancers
        - Servers can be rebooted, taken down for upgrade, additional servers can be added, etc., all transparently to the users.
    - Spring Cloud Configuration server
        - Allows configurations for development, test, and production to be maintained in the same location and distributed across all servers.
    - Distributed service registry
        - Keeps registered services synchronized across all servers.
    - Distributed ticket registry
        - Distributes tickets across all back-end servers so that any server can service a client, even when servers go down or restart.
- Support for additional protocols
    - Built-in support for SAML 2.0 IdP and SP
        - No more Shibboleth servers!
        - Support for many SPs built in: Adobe Creative Cloud, Google Apps, Office 365, Tableau, Workday, …
    - Built-in support for multi-factor authentication
        - Duo Security (forthcoming for faculty and staff)
        - Google Authenticator (perhaps, for students)
- New management console and management webapps
    - Management console
        - Dashboard for monitoring server status and performance
        - Active sessions and authentications
        - Metrics and statistics
    - Services management
        - Add, edit, delete, enable, disable services

        - Attribute release, access rules, etc.
- Other interesting features (for experimentation)
    - Risk-based authentications
    - Password management

Although the Apereo development team has dramatically simplified the configure-build-deploy process, CAS 5 is still a complex system with a lot of moving parts, and there can be a pretty steep learning curve for someone who's never done it before. Since there's not a lot of up-to-date step-by-step how-to documentation out there, we're offering what we've learned in the hope that others will find it helpful.

> **DISCLAIMER**
>
> The instructions and settings provided in this document may not be the only way to do things. They may not be the best way to do things. They may not even be the right way to do things. They work for us, but they may not work for you. You should carefully evaluate every suggestion, recommendation, and instruction in the context of your environment and decide whether or not it makes sense. Make sure you know and understand what's going to happen ***before*** you press the "Enter" key. When in doubt, Read The Fine Manual .
>
> No warranty express or implied. May cause drowsiness. Your mileage may vary. Not intended to diagnose, treat, cure, or prevent any disease. Professional driver on closed course. Safety goggles recommended. Use with adult supervision. Keep out of reach of children. Do not eat.

SSO environment architecture

The New School's dependence on its single sign-on environment continues to grow, making the availability of the environment ever more important. The CAS 5 server environment will be designed for high availability by ensuring that each component of the environment is sufficiently redundant to make the service resilient to multiple component failures and to enable routine maintenance on the environment to be performed without incurring service downtime. Figure 1 below highlights the principal aspects of this design.

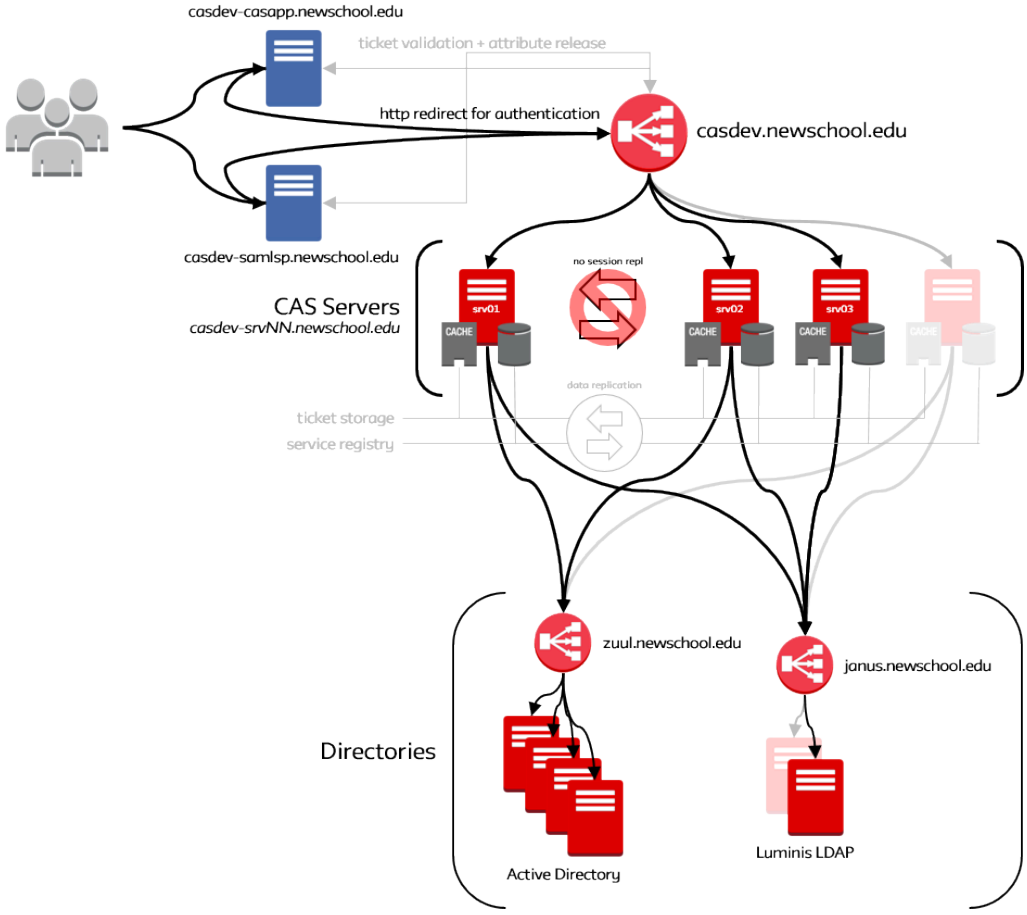

Figure 1. The New School CAS 5 development environment

Starting with the red circle near the top of the figure, the user- (or client-) facing domain name for the single sign-on service will resolve to a virtual address on the F5 load balancers. In the development environment, this domain name will be ***casdev.newschool.edu***. The F5s are deployed in a high availability configuration between our two primary data centers, and fail over automatically in the event the primary unit goes down.

Multiple CAS servers will be deployed in an active-active configuration. A distributed ticket registry (cache) that replicates all tickets to all servers will be used to ensure that a ticket can be located from any server (the server that is asked to validate a ticket may not be the same server that originally created it). A distributed service registry that replicates all registered services to all servers will be used to ensure that all servers support the same set of services. And finally, a configuration server will be used to ensure that server configuration settings are kept in sync across all the servers. To eliminate the need to replicate Java servlet container sessions across servers (a complex and unreliable process), session affinity will be enabled on the F5s. This option tells the F5s to remember which server a new client is first directed to, and direct all requests from that client to that server for a short period of time.

The production environment will require a minimum of four CAS servers, two in each primary data center, to ensure that redundancy is maintained even if one data center goes offline. However, as shown in the figure, the development environment (as well as the test environment) can get by with just three servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03.newschool.edu***), which conserves VMware resources while still enabling us to validate replication operations in the more complex “N > 2” case.

Two user directories are used to provide authentication services: Active Directory and Luminis LDAP. The Active Directory environment is already configured for high availability, with two domain controllers in each data center (and a fifth one in Paris) fronted by a virtual address (***zuul.newschool.edu***) on the F5s. The Luminis LDAP environment is not currently configured for high availability; there is only a single server instance. However, the instance is located behind a virtual address (***janus.newschool.edu***) on the F5s, and the underlying technology (OpenDJ) supports replication, so enabling high availability should be relatively straight forward (doing so is outside the scope of this document, however).

To facilitate development and testing, the development environment will also include two single sign-on enabled applications. Shown at the top left of Figure 1 as blue boxes, ***casdev-casapp.newschool.edu*** will be a CAS-enabled Apache web server, and ***casdev-samlsp.newschool.edu*** will be a SAML2-enabled Apache web server.

References

- CAS 5: High Availability Guide (HA/Clustering)

Leveraging the cloud (future)

Note: The environment described below represents future work that is outside the scope of this document. We describe it here to record our thinking on the topic, but we have no plans to implement a hybrid environment at the present time.

The on-premises single sign-on environment described in the previous section, even though designed for high availability, may still be vulnerable to large-scale adverse events such as natural disasters (Hurricane Sandy, 2012), public infrastructure failures (Northeast power outage, 2003), or Tier-1 Internet provider outages (Level3 Communications, 2013). If such an event were to occur and “take out” both New School data centers or both New School Internet connections, the impact would be felt across more than just the other New School applications hosted in our on-premises data centers. Cloud-hosted applications, such as Google G Suite, Workday, and Canvas would also become unavailable, because they depend on the New School single sign-on service to enable users to log in.

Figure 2 shows how Amazon Web Services (AWS) could be used to add a cloud-based component to the on-premises deployment described in the previous section. In this hybrid design, on-campus users would continue to use the on-premises server environment to authenticate, providing better performance, while off-campus users would use the cloud-based environment.

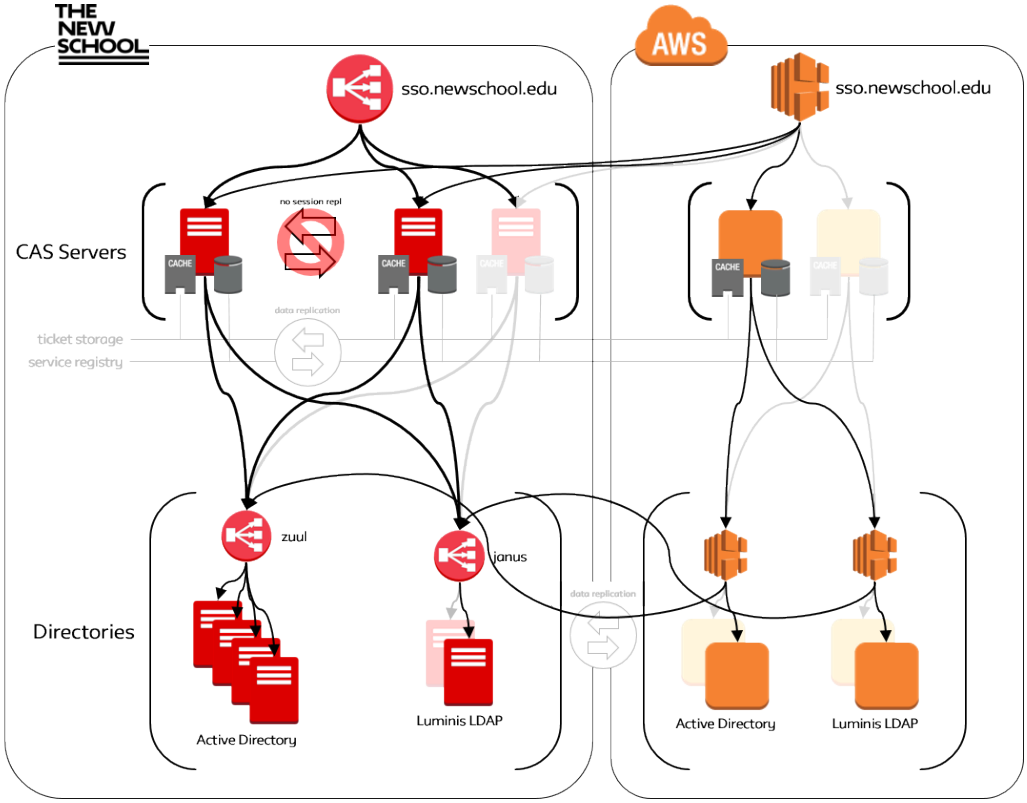

Figure 2. Hybrid on-premises/cloud CAS 5 environment

To keep on-campus users authenticating against the on-premises server environment while directing off-campus users to the hybrid environment, the New School domain name servers would be configured such that the internal servers (used by on-campus devices) and external servers (used by off-campus devices) return different results when resolving the user- (or client-) facing domain name for the single sign-on service. The internal name servers would continue to resolve the domain name to a virtual address on the F5 load balancers, resulting in on-campus users being directed to the on-premises environment as described previously. The external name servers, however, would resolve the domain name to a virtual address on an AWS Elastic Load Balancer.

The user-facing Elastic Load Balancer (at the top of the diagram) could be a single instance or, to ensure high availability, multiple instances (perhaps even across multiple availability zones). As shown in the diagram, the load balancer would route connections to a mixed pool of on-premises and cloud-based CAS servers. Utilizing the on-premises servers reduces the number of cloud-based servers needed, although there must still be enough cloud-based servers to ensure availability and response time when the on-premises environment is unavailable. The cloud-based CAS servers would be configured to be members of the same ticket storage and service registry replication pools as the on-premises servers, to ensure a seamless user experience regardless of which servers are accessed.

The cloud-based environment would also contain cloud-based instances of Active Directory and Luminis LDAP. Additional Elastic Load Balancers would be used to route directory queries to mixed pools of on-premises and cloud-based directory servers. As above, utilizing the on-premises servers reduces the number of cloud-based servers (and the amount of storage) needed, although there must still be enough cloud-based servers of each type to ensure availability and response time when the on-premises environment is unavailable. The cloud-based servers would be configured to replicate with their on-premises counterparts, ensuring a seamless user experience regardless of which servers are accessed.

The addition of a cloud-based component to the on-premises deployment described in the previous section will ensure that the New School single sign-on service is resilient even in the face of large-scale adverse events that “take out” the on-premises environment. The degree of resilience provided by the cloud environment can be increased or decreased through the deployment of additional server instances, the use of multiple availability zones, or even the use of multiple cloud providers.

Setting up the environment

Summary: Before beginning the CAS build and configuration process, the server environment should be prepared by creating virtual machines, installing necessary software dependencies, and performing basic software configuration and system administration tasks.

The New School's CAS 5 development environment is comprised of six servers, all in the ***newschool.edu*** domain. There is one master build server:

casdev-master 192.168.100.100	The master build server where software will be built for deployment to the other servers. This server will include development tools (compilers, libraries, etc.) that are not appropriate for installation on user-facing servers.

There is also a pool of three identical CAS servers:

casdev-srv01 192.168.100.101	A CAS server instance; a member of the F5 load balancers' server pool for the ***casdev.newschool.edu*** virtual address.
casdev-srv02 192.168.100.102	A CAS server instance; a member of the F5 load balancers' server pool for the ***casdev.newschool.edu*** virtual address.
casdev-srv03 192.168.100.103	A CAS server instance; a member of the F5 load balancers' server pool for the ***casdev.newschool.edu*** virtual address.

And there are two sample client application servers:

casdev-casapp 192.168.100.201	A user-facing client application (Apache web server) used to test the CAS protocol and attribute release.
casdev-samlsp 192.168.100.202	A user-facing client application (Apache web server) used to test the SAML 2.0 protocol and attribute release.

The environment also includes a single virtual address on the F5 load balancers (also in the ***newschool.edu*** domain):

casdev 192.168.200.10	User-facing domain name and virtual address that manages access to the pool of CAS servers (***casdev-srvNN***) to provide load balancing, high availability, and fault tolerance.

Each of the six development servers is a VMware virtual machine running Red Hat Enterprise Linux (RHEL) 7 (64-bit) on 1 CPU with 4 GB of RAM and 20 GB of disk space.

> ⚠ **Important:** Although 1 CPU and 4 GB is adequate for CAS development and testing, production operation requires more resources. Each server in the production server pool should have a minimum of 2 CPUs and 8 GB of memory.

References

- CAS 5: Installation Requirements

Initial setup tasks

Before starting the process of configuring the various servers to perform their individual roles in the development environment, there are some initial setup tasks to be performed.

Ensure that all systems are up-to-date

It's important to make sure that the operating system software on the servers is up-to-date. Run the command

```
# yum -y update
```

on each of the six servers in the environment to ensure that all software installed on the base system is up-to-date. If running this command results in updates to system shared libraries or the operating system kernel, reboot the server(s) before continuing.

Install development tools on the master build server

The master build server (***casdev-master***) will be used to build and compile software from source code. Run the command

```
casdev-master# yum -y groupinstall "Development Tools"
```

to install the tools needed to do that. This command should only be run on the master build server (***casdev-master***); it is not necessary (or desirable) to install these tools on any of the other servers.

Install Perl test modules on the master build server

Some of the software packages to be installed use Perl testing modules to perform their tests. Run the commands

```
casdev-master# yum -y install perl-Module-Load-Conditional
casdev-master# yum -y install perl-Test-Simple
```

to install the necessary modules. This command should only be run on the master build server (***casdev-master***); it is not necessary to install these modules on any of the other servers.

Configure Git (optional)

The CAS project team uses GitHub to host all of its code, maintain version control, and allow collaboration among developers. Once we start Building the CAS server (page 86), we'll be using Git commands to make local copies of files from the CAS GitHub repositories, to track our changes and additions to those files as we customize them, and to keep our local files in sync with any corrections and updates made to the master copies by the project team.

> ☑ **Tip:** If you're unfamiliar with Git and GitHub, you may want to read *Pro Git* (chapters 2, 3, and 6).

When making a Git commit (recording a change to a file as a new version), Git requires that the user name and email address of the person making the commit be provided so they can be recorded in the commit history. To avoid having to specify these values every time, they can be configured ahead of time by running the commands

```
casdev-master# git config --global user.name "David A. Curry"
casdev-master# git config --global user.email "david.curry@newschoo
l.edu"
```

on the master build server (***casdev-master***). (Substitute your name and email address for the values inside the quotation marks.) When making a commit, Git will also ask for some text to describe what was changed, and will invoke a text editor to allow that text to be entered. Run the command

```
casdev-master# git config --global core.editor "vim"
```

to configure the editor that will be used for this purpose (the example above sets the editor to `vim`; another common choice is `emacs`).

Set up SSH public key authentication (optional)

To make it easier to distribute the software built on ***casdev-master*** to the other servers in the environment, we will create a public/private authentication key pair that will allow ***casdev-master*** to connect to those servers via `ssh` and `scp` without a password. Run the command

```
casdev-master# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
89:e0:9e:11:78:64:64:22:d3:df:74:30:38:e3:ec:c6 root@casdev-master.ne
wschool.edu
The key's randomart image is:
+--[ RSA 2048]----+
|o...=.o.         |
| o.*+ ...        |
|  .++= .         |
|   o+o.. .       |
|   oo . S        |
|   .Eo           |
|   .o            |
|                 |
|                 |
+-----------------+
casdev-master#
```

on the master build server (***casdev-master***) to generate the key pair. Once the key pair has been generated, run the command

```
casdev-master# ssh-copy-id casdev-srv01.newschool.edu
The authenticity of host 'casdev-srv01.newschool.edu (192.168.20.1)'
can't be established.
ECDSA key fingerprint is 43:51:43:a1:b5:fc:8b:b7:0a:3a:a9:b1:0f:66:7
3:a8.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new ke
y(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if yo
u are prompted now it is to install the new keys
root@casdev-srv01.newschool.edu's password: (enter remote server pass
word)

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'casdev-srv01.newschoo
l.edu'"
and check to make sure that only the key(s) you wanted were added.
casdev-master#
```

to copy it to ***casdev-srv01***. Repeat the `ssh-copy-id` step to copy the key to each of the other servers in the environment (***casdev-srv02***, ***casdev-srv03***, ***casdev-casapp***, and ***casdev-samlsp***).

Configure time synchronization

For an active-active, multiple-server environment such as the one we're building to work properly, the time-of-day clocks on all servers in the environment must be in agreement. The Network Time Protocol (NTP) is used to ensure that each server is synchronized to Coordinated Universal Time (UTC).

> ℹ **Note:** The steps in this section should be performed on all six servers in the environment.

Determine if NTP is already in use

RHEL 7 offers two NTP implementations, `ntpd` and `chronyd`. Run the commands

```
# systemctl status chronyd
# systemctl status ntpd
```

to determine whether either of these is already in use. If output similar to this:

```
● chronyd.service - NTP client/server
    Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled)
    Active: active (running) since Ddd YYYY-MM-DD HH:MM:SS EDT; 58s a
go
Main PID: 2530 (chronyd)
    CGroup: /system.slice/chronyd.service
            └─2530 /usr/sbin/chronyd -u chrony
```

or this:

```
● ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; ven
dor preset: disabled)
   Active: active (running) since Ddd YYYY-MM-DD HH:MM:SS EDT; 58s ag
o
  Process: 25086 ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS (code=e
xited, status=0/SUCCESS)
 Main PID: 25087 (ntpd)
   CGroup: /system.slice/ntpd.service
           └─25087 /usr/sbin/ntpd -u ntp:ntp -g
```

appears, then one service or the other is installed and running, and nothing further needs to be done (go to the next section, Install Apache Tomcat on the CAS servers (page 25)). On the other hand, if output similar to this:

```
● ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; disabled; ve
ndor preset: disabled)
   Active: inactive (dead)
```

or this:

```
Unit chronyd.service could not be found.
```

appears for both commands, then time is not being synchronized and NTP needs to be installed on each server in the development environment by following the remaining steps in this section.

Install NTP (`ntpd`)

Generally, `ntpd` is preferred for always-on systems like servers, while `chronyd` is intended for use on systems like laptops that are shut down frequently or connected only intermittently to a network. Run the command

```
# yum -y install ntp
```

to install `ntpd`.

Configure `/etc/ntp.conf`

Edit the file `/etc/ntp.conf` and replace its entire contents with the `ntpd` configuration used by your organization. If your organization doesn't have a standard configuration, use something like the following example, which makes use of public time servers from the NTP Pool Project:

```
#
# Network Time Protocol configuration file (/etc/ntp.conf)
#
# Use this configuration file on Stratum 3 Linux systems.
#

#
# Stratum 2 servers. The total number of servers listed should be at
least 2 more
# than the number specified for minclock and minsane, below.
#
server          0.us.pool.ntp.org               iburst
server          1.us.pool.ntp.org               iburst
server          2.us.pool.ntp.org               iburst
server          3.us.pool.ntp.org               iburst
server          0.north-america.pool.ntp.org    iburst
server          1.north-america.pool.ntp.org    iburst
server          2.north-america.pool.ntp.org    iburst
server          3.north-america.pool.ntp.org    iburst

#
# At least minsane candidate servers must be available for selectio
n, and the
# mitigation algorithm must produce at least minclock candidates. Byz
antine
# agreement principles require at least 4 candidates to correctly dis
card a
# single falseticker.
#
# http://support.ntp.org/bin/view/Support/StartingNTP4#Sectio
n_7.1.4.3.1.
#
tos minsane     4
tos minclock    4

#
# File used to record the frequency of the local clock oscillator. Th
is is
# used at startup to set the initial frequency.
#
driftfile       /var/lib/ntp/drift
```

If you're not in the United States or North America, consult the NTP Pool Project's pool server lists for a list of servers in your country or on your continent.

Open the NTP port in the firewall

In order to synchronize time with other systems, `ntpd` needs to be able to communicate on UDP port 123. RHEL 7's `firewalld` includes a pre-defined service for `ntp` to enable this. Run the commands

```
# firewall-cmd --zone=public --add-service=ntp --permanent
success
# firewall-cmd --reload
success
#
```

to open that service in the system firewall.

Enable and start `ntpd`

Run the commands

```
# systemctl enable ntpd
Created symlink from /etc/systemd/system/multi-user.target.wants/ntp
d.service to /usr/lib/systemd/system/ntpd.service.
# systemctl start ntpd
```

to enable and start `ntpd`. After about 15-20 minutes, the protocol should have selected a server to synchronize with, and its status can be checked with the `ntpstat` and/or `ntpq` commands:

```
# ntpstat
synchronised to NTP server (208.75.89.4) at stratum 3
   time correct to within 72 ms
   polling server every 1024 s
# ntpq -p localhost
     remote           refid      st t when poll reach   delay   offse
t  jitter
==============================================================================
=========
+ns20.alltraders 127.67.113.92    2 u  713 1024  377   82.958   -2.84
9   1.783
-four10.gac.edu  18.26.4.105      2 u  403 1024  377   35.805   -0.44
4   2.610
-barry.tsi.io    198.60.22.240    2 u  352 1024  377   88.362   -0.08
8   2.111
-mdnworldwide.co 127.67.113.92    2 u  412 1024  371   52.765    2.50
9   4.455
-static-96-244-9 192.168.10.254   2 u  268 1024  377   10.932    1.25
5   3.564
+srcf-ntp.stanfo 171.64.7.105     2 u  219 1024  377   82.896   -3.09
2   1.710
*time.tritn.com  198.60.22.240    2 u  530 1024  377   68.547   -1.66
0   3.617
+bindcat.fhsu.ed 132.163.4.103    2 u  916 1024  377   58.940   -2.84
1   2.145
#
```

Install Apache Tomcat on the CAS servers

Summary: Apache Tomcat will be used as the Java Servlet container for CAS. To ensure a best practices security configuration, the latest versions of Java, Tomcat, OpenSSL, and the Tomcat Native Library will be installed on the master build server and then distributed to the CAS servers.

Note: CAS 5 is based on Spring Boot, which means that instead of building and installing an external servlet container, CAS can be deployed using an embedded servlet container and invoked with a `java -jar cas.war` style command. However, because we will be deploying multiple Java applications on the same set of servers, we will use the more traditional external container approach. This will avoid network port conflicts, allow us to make more efficient use of system resources, and enable us to manage our CAS installation in the same way we manage our other Java servlet-based applications.

CAS 5 requires Java 8 and a Java Servlet container that supports v3.1 or later of the Java Servlet specification. Apache Tomcat is the most commonly used container for CAS, so that's what we'll use for this deployment. Tomcat 8.0 was the first version to support v3.1 of the servlet specification, but this version has been superseded by Tomcat 8.5. Unfortunately, Red Hat does not, as of this writing, offer Tomcat 8.5 on RHEL 7, so it will have to be installed manually.

The *CAS Security Guide* warns that all communication with the CAS server must occur over a secure channel (i.e., Transport Layer Security) to prevent the disclosure of users' security credentials and/or CAS ticket-granting tickets. From a practical standpoint this means that all CAS URLs must use HTTPS, but it also means that all connections from the CAS server to the calling application must use HTTPS as well. Consequently, it's critical that Tomcat's TLS settings be configured in line with recognized best practices for cipher suite choice, key lengths, forward secrecy, and other parameters.

In the past, it has been difficult to configure Tomcat according to TLS best practices because of limitations in both Tomcat and the Java Secure Socket Extension (JSSE). Java 8 and Tomcat 8.5 have made significant improvements in this area, but installation of the optional Tomcat Native Library, which uses OpenSSL instead of Java cryptography libraries, is still the best way to ensure that a Tomcat installation scores an 'A' on the Qualys® SSL Labs SSL Server Test. The version of

the Tomcat Native Library included with Tomcat 8.5 requires OpenSSL 1.0.2 or higher. Unfortunately, Red Hat does not, as of this writing, offer OpenSSL 1.0.2 or higher on RHEL 7, so it will also have to be installed manually.

References

- CAS 5: Security Guide
- CAS 5: Servlet Container Configuration
- Tomcat Wiki: TLS Cipher suite choice

Install an entropy daemon

> ℹ **Note:** This step is recommended if running CAS on virtual Linux servers. It is not necessary if running CAS on physical Linux servers or Windows servers of either type.

A common problem on virtual Linux servers is that the `/dev/random` device will run low on entropy, because most of the sources the kernel uses to build up the entropy pool are hardware-based, and therefore do not exist in a virtual environment. If there's not enough entropy available when Tomcat is started, it can often take two or three minutes or longer for the server to start. Once Tomcat has started and the CAS application has been loaded, entropy is still required to establish secure (HTTPS) connections with authenticating users' browsers and protected applications. A lack of available entropy will adversely affect the performance of the application by limiting the rate at which connections can be processed.

To improve the size of the entropy pool on Linux, it's possible to feed random data from an external source into `/dev/random`. One way to do this is the `haveged` daemon, which uses the HAVEGE (HArdware Volatile Entropy Gathering and Expansion) algorithm to harvest the indirect effects of hardware events on hidden processor state (caches, branch predictors, memory translation tables, etc) to generate random bytes with which to fill `/dev/random` whenever the supply of random bits falls below the low water mark of the device. We will use this approach to avoid entropy depletion on the CAS servers.

Red Hat does not offer `haveged` on RHEL 7, but it can be installed from the Fedora Project's Extra Packages for Enterprise Linux (EPEL) repository.

Install the EPEL repository

Run the commands

```
# cd /tmp
# curl -LO https://dl.fedoraproject.org/pub/epel/epel-release-lates
t-7.noarch.rpm
  % Total    % Received % Xferd  Average Speed   Time    Time     Tim
e  Current
                                 Dload  Upload   Total   Spent    Lef
t  Speed
100 14848  100 14848    0     0  27561      0 --:--:-- --:--:--
--:--:-- 27547
# yum -y install epel-release-latest-7.noarch.rpm
# rm -f epel-release-latest-7.noarch.rpm
```

on the master build server (***casdev-master***) and the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) to install the EPEL repository. It is not necessary to install the EPEL repository on the client application servers (***casdev-casapp*** and ***casdev-samlsp***).

Install haveged

Run the commands

```
# yum -y install haveged
# systemctl enable haveged
Created symlink from /etc/systemd/system/multi-user.target.wants/have
ged.service to /usr/lib/systemd/system/haveged.service.
# systemctl start haveged
```

on the master build server (***casdev-master***) and the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) to install, enable, and start `haveged`. It is not necessary to install `haveged` on the client application servers (***casdev-casapp*** and ***casdev-samlsp***).

References

- haveged—a simple entropy daemon
- HAVEGE project

Install Java

CAS 5 requires Java 8 or higher. You can use either the OpenJDK or the Oracle version of the Java Development Kit (JDK), but if you opt for the Oracle version, you will also have to install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files to enable the use of stronger cryptographic algorithms and longer keys. Other than that, there is little practical difference between the two from CAS' perspective.

Since OpenJDK doesn't require us to download and maintain yet another package, we'll use that version for this deployment. Run the command

```
# yum -y install java-1.8.0-openjdk-devel
```

on the master build server (***casdev-master***) and the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) to install OpenJDK Java 8. It is not necessary (or desirable) to install Java on the client application servers (***casdev-casapp*** and ***casdev-samlsp***).

Because it's possible to have more than one version of Java installed at the same time, run the command

```
# java -version
openjdk version "1.8.0_141"
OpenJDK Runtime Environment (build 1.8.0_141-b16)
OpenJDK 64-Bit Server VM (build 25.141-b16, mixed mode)
#
```

to check that Java 8 is indeed the default version (the version number should start with "1.8"; the remainder will vary depending on what the current patch level is).

Install Tomcat

> ℹ **Note:** The steps in this and the next several sections should only be performed on the build server (***casdev-master***). After everything has been built, configured, and tested, the installation will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***).

As discussed in the introduction to this section (page 25), Red Hat does not offer Tomcat 8.5 on RHEL 7, so it must be downloaded and installed manually. Run the commands

```
casdev-master# mkdir -p /opt/tomcat
casdev-master# cd /opt/tomcat
casdev-master# curl -LO 'http://apache.cs.utah.edu/tomcat/tomcat-8/v
8.5.27/bin/apache-tomcat-8.5.27.tar.gz'
  % Total    % Received % Xferd  Average Speed   Time    Time     Tim
e  Current
                                 Dload  Upload   Total   Spent    Lef
t  Speed
100 9313k  100 9313k    0     0  4036k      0  0:00:02  0:00:02
--:--:-- 4038k
casdev-master# tar xzf apache-tomcat-8.5.27.tar.gz
casdev-master# ln -s apache-tomcat-8.5.27 latest
casdev-master# rm -f apache-tomcat-8.5.27.tar.gz
```

to download and install Tomcat on the master build server (***casdev-master***). (Replace `8.5.27` in the commands above with the current stable version of Tomcat 8.5.) This will install the current version of Tomcat 8.5 in a version-specific subdirectory of `/opt/tomcat`, and make it accessible by the path `/opt/tomcat/latest`. By making everything outside this directory refer to `/opt/tomcat/latest`, an updated version can be installed and the link changed without having to edit/recompile anything else.

Install Tomcat dependencies

As discussed in the introduction to this section (page 25), the Tomcat Native Library is the best way to ensure that Tomcat meets TLS best practices. The library is included with Tomcat 8.5, but must be compiled and installed manually. To do that, the current versions of OpenSSL and the Apache Portable Runtime, which are not offered by Red Hat for RHEL 7, must also be downloaded, compiled, and installed. Lastly, we will also compile and install the Apache Commons Daemon to help with installing and running Tomcat as a system service.

All of the steps in this section should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

OpenSSL

The version of the Tomcat Native Library included with Tomcat 8.5 requires OpenSSL 1.0.2 or higher, which is not offered by Red Hat on RHEL7. Run the commands

```
casdev-master# cd /tmp
casdev-master# curl -LO 'https://www.openssl.org/source/openssl-1.1.0
g.tar.gz'
  % Total    % Received % Xferd  Average Speed   Time    Time     Tim
e  Current
                                 Dload  Upload   Total   Spent    Lef
t  Speed
100 5278k  100 5278k    0     0  33.4M      0 --:--:-- --:--:--
--:--:-- 33.4M
casdev-master# tar xzf openssl-1.1.0g.tar.gz
casdev-master# cd openssl-1.1.0g
casdev-master# mkdir -p /opt/openssl/openssl-1.1.0g
casdev-master# ln -s openssl-1.1.0g /opt/openssl/latest
casdev-master# ./config --prefix=/opt/openssl/openssl-1.1.0g shared
(lots of output... check for errors)
casdev-master# make depend
casdev-master# make
(lots of output... check for errors)
casdev-master# make test
(lots of output... check for errors)
casdev-master# make install_sw
(lots of output... check for errors)
casdev-master# cd /tmp
casdev-master# rm -rf openssl-1.1.0g openssl-1.1.0g.tar.gz
```

to download and build OpenSSL on the master build server (***casdev-master***). (Replace `1.1.0g` in the commands above with the current stable version of OpenSSL.)

> ⚠ **Important:** The `INSTALL` document in the OpenSSL source directory says that tests **must** be run as an unprivileged user. If you decide to ignore this and run them as `root` anyway, you should expect the `40-test_rehash.t` test to fail.

This will install the current version of OpenSSL in a version-specific subdirectory of `/opt/openssl`, and make it accessible by the path `/opt/openssl/latest`. By linking the Tomcat Native Library against `/opt/openssl/latest`, an updated version of OpenSSL can be installed and the link changed without having to recompile anything else.

Apache Portable Runtime

The Tomcat Native Library also depends on the Apache Portable Runtime (APR) library. Although the version of the APR library provided by Red Hat with RHEL7 is compatible with the Tomcat Native Library included with Tomcat 8.5, it’s several versions behind the current release. Since we’re building other dependencies anyway, we’ll build this one too, just for completeness. Run the commands

```
casdev-master# cd /tmp
casdev-master# curl -LO 'http://apache.cs.utah.edu//apr/apr-1.6.3.ta
r.gz'
% Total    % Received % Xferd  Average Speed   Time    Time     Tim
e  Current
                                Dload  Upload   Total   Spent    Lef
t  Speed
100 1045k  100 1045k    0     0   439k      0  0:00:02  0:00:02
--:--:--  439k
casdev-master# tar xzf apr-1.6.3.tar.gz
casdev-master# cd apr-1.6.3
casdev-master# mkdir -p /opt/apr/apr-1.6.3
casdev-master# ln -s apr-1.6.3 /opt/apr/latest
casdev-master# ./configure --prefix=/opt/apr/apr-1.6.3
casdev-master# make
(lots of output... check for errors)
casdev-master# make test
(lots of output... check for errors)
casdev-master# make install
(lots of output... check for errors)
casdev-master# cd /tmp
casdev-master# rm -rf apr-1.6.3 apr-1.6.3.tar.gz
```

to download and build the APR library on the master build server (***casdev-master***). (Replace `1.6.3` in the commands above with the current stable version of the APR library.)

This will install the current version of the APR library in a version-specific subdirectory of `/opt/apr`, and make it accessible by the path `/opt/apr/latest`. By linking the Tomcat Native Library against `/opt/apr/latest`, an updated version of the APR library can be installed and the link changed without having to recompile anything else.

Tomcat Native Library

The Tomcat Native Library source code is included as part of the Tomcat distribution; it just needs to be extracted, compiled, and installed. Run the commands

```
casdev-master# cd /opt/tomcat/apache-tomcat-8.5.27/bin
casdev-master# tar xzf tomcat-native.tar.gz
casdev-master# cd tomcat-native-*-src/native
casdev-master# ./configure \
  --with-java-home=/usr/lib/jvm/java-openjdk \
  --with-apr=/opt/apr/latest/bin/apr-1-config \
  --with-ssl=/opt/openssl/latest \
  --prefix=/opt/tomcat/apache-tomcat-8.5.27
(lots of output... check for errors)
casdev-master# make
(lots of output... check for errors)
casdev-master# make install
(lots of output... check for errors)
casdev-master# cd ../..
casdev-master# rm -rf tomcat-native-*-src
```

to build the Tomcat Native Library on the master build server (***casdev-master***). (Replace `8.5.27` in the commands above with the version of Tomcat installed earlier.)

This will install the Tomcat Native Library in the `lib` directory of its associated Tomcat installation.

Apache Commons Daemon (`jsvc`)

The Apache Commons Daemon (`jsvc`) allows Tomcat to be started as `root` to perform some privileged operations (such as binding to ports below 1024) and then switch identity to run as a non-privileged user, which is better from a security perspective. The daemon is included as part of the Tomcat distribution; it just needs to be extracted, compiled, and installed. Run the commands

```
casdev-master# cd /opt/tomcat/apache-tomcat-8.5.27/bin
casdev-master# tar xzf commons-daemon-native.tar.gz
casdev-master# cd commons-daemon-*-native-src/unix
casdev-master# ./configure --with-java=/usr/lib/jvm/java-openjdk
(lots of output... check for errors)
casdev-master# make
(lots of output... check for errors)
casdev-master# mv jsvc ../..
casdev-master# cd ../..
casdev-master# rm -rf commons-daemon-*-native-src
```

to build the Tomcat Native Library on the master build server (***casdev-master***). (Replace `8.5.27` in the commands above with the version of Tomcat installed earlier.)

This will install the `jsvc` program in the bin directory of its associated Tomcat installation.

Organize the installation

To make it easier to upgrade Tomcat from one version to another without having to reapply configuration files changes or reinstall web applications, it makes sense to move these parts of the Tomcat installation to different parts of the file system and install symbolic links in their place.

All of the commands in this section should be run on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

Move the `conf` directory to `/etc/tomcat`

The `conf` subdirectory contains Tomcat's configuration files. Although these files can change from one major version to another, they don't typically have to change when installing newer minor versions. Run the commands

```
casdev-master# cd /opt/tomcat/latest
casdev-master# cp -rp conf /etc/tomcat
casdev-master# rm -rf conf
casdev-master# ln -s /etc/tomcat conf
```

to move Tomcat's configuration files to `/etc/tomcat`.

Move the `logs` directory to `/var/log/tomcat`

Log files can grow very large; storing them in `/opt` is not a good practice. Since log files are typically stored in `/var/log` on Linux systems, it makes sense to store Tomcat's log files there as well. Run the commands

```
casdev-master# cd /opt/tomcat/latest
casdev-master# cp -rp logs /var/log/tomcat
casdev-master# rm -rf logs
casdev-master# ln -s /var/log/tomcat logs
```

to move Tomcat's log files to `/var/log/tomcat`.

Move the `webapps` directory to `/var/lib/tomcat`

Although Tomcat runs web applications, the web applications themselves are not part of Tomcat. Therefore, it doesn't make a lot of sense to keep them inside the Tomcat installation directory, where they will have to be reinstalled every time Tomcat is updated. Run the commands

```
casdev-master# cd /opt/tomcat/latest
casdev-master# cp -rp webapps /var/lib/tomcat
casdev-master# rm -rf webapps
casdev-master# ln -s /var/lib/tomcat webapps
```

to move the `webapps` directory to `/var/lib/tomcat`.

Move the `work` directory to `/var/cache/tomcat/work`

Tomcat's `work` directory is where translated servlet source files and JSP/JSF classes are stored. Its contents are created automatically, but don't need to be recreated unless the application has been changed. To reduce startup time, the contents of this directory should be preserved across application restarts and system reboots. Linux systems provide the `/var/cache` directory for just that purpose, so we can put the `work` directory there. Run the commands

```
casdev-master# cd /opt/tomcat/latest
casdev-master# mkdir /var/cache/tomcat
casdev-master# cp -rp work /var/cache/tomcat/work
casdev-master# rm -rf work
casdev-master# ln -s /var/cache/tomcat/work work
```

to move the `work` directory to `/var/cache/tomcat/work`. Note that we created a `work` subdirectory in `/var/cache/tomcat`; this is so that we can also use `/var/cache/tomcat` to store Tomcat's `temp` directory (see below).

Move the `temp` directory to `/var/cache/tomcat/temp`

Tomcat provides a `temp` directory for web applications to store temporary files in. But like log files, temporary files can sometimes be very large, so storing them in `/opt` is probably not a good practice. But `/tmp` and `/var/tmp` are not the best places either, because we want to be able to limit access to Tomcat's temporary files (see Harden the installation (page 39)). Therefore, we will create a new `temp` directory under `/var/cache/tomcat`. Run the commands

```
casdev-master# cd /opt/tomcat/latest
casdev-master# cp -rp temp /var/cache/tomcat/temp
casdev-master# rm -rf temp
casdev-master# ln -s /var/cache/tomcat/temp temp
```

to move Tomcat's `temp` directory to `/var/cache/tomcat/temp`.

Harden the installation

The *Tomcat Security Considerations* document makes several recommendations for hardening a Tomcat installation:

- Tomcat should not be run as the `root` user; it should be run as a dedicated user (usually named `tomcat`) that has minimum operating system permissions. It should not be possible to log in remotely as the `tomcat` user.
- All Tomcat files should be owned by user `root` and group `tomcat` (the `tomcat` user's default group should be group `tomcat`). File/directory permissions should be set to owner read/write, group read only, and world none. The exceptions are the `logs`, `temp`, and `work` directories, which should be owned by the `tomcat` user instead of `root`.
- The default and example web applications included with the Tomcat distribution should be removed if they are not needed.
- Auto-deployment should be disabled, and web applications should be deployed as exploded directories rather than web application archives (WAR files).

Implementing these recommendations means that, even if an attacker compromises the Tomcat process, he or she cannot change the Tomcat configuration, deploy new web applications, or modify existing web applications.

All of the steps in this section should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

Create a `tomcat` user and `tomcat` group

Run the commands

```
casdev-master# groupadd -r tomcat
casdev-master# useradd -r -d /opt/tomcat -g tomcat -s /sbin/nologin t
omcat
```

to create a `tomcat` user and a `tomcat` group.

Set file ownership and permissions

Run the commands

```
casdev-master# mkdir -p /opt/tomcat/latest/conf/Catalina/localhost
casdev-master# for dir in . conf webapps
> do
> cd /opt/tomcat/latest/$dir
> chown -R root.tomcat .
> chmod -R u+rwX,g+rX,o= .
> chmod -R g-w .
> done
casdev-master# for dir in logs temp work
> do
> cd /opt/tomcat/latest/$dir
> chown -R tomcat.tomcat .
> chmod -R u+rwX,g+rX,o= .
> chmod -R g-w .
> done
casdev-master#
```

to set the proper file ownerships and permissions. Note that, as discussed above, some of the directories are owned by `root`, while others are owned by `tomcat`.

Remove example webapps

Run the commands

```
casdev-master# cd /opt/tomcat/latest
casdev-master# rm -rf temp/* work/*
casdev-master# cd webapps
casdev-master# rm -rf docs examples host-manager manager
```

to remove unneeded example web applications.

> ⚠ **Important:** The command above does not remove the `ROOT` web application from the `webapps` directory because it can be useful in a development/test environment to quickly determine whether Tomcat is working properly. However, when deploying Tomcat to production servers, the `ROOT` application should be removed along with the rest of the default web applications.

Disable auto-deployment

To disable auto-deployment, edit the file `/opt/tomcat/latest/conf/server.xml` and locate the `Host` XML tag (around line 148), which should look something like this:

```
<Host name="localhost"  appBase="webapps"
      unpackWARs="true" autoDeploy="true">
```

Disable application auto-deployment and unpacking of web application archive files by setting these attributes to `false`:

```
<Host name="localhost"  appBase="webapps"
      unpackWARs="false" autoDeploy="false">
```

References

- Tomcat Security Considerations

Configure TLS/SSL settings

All of the servers in the load balancer server pool (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) will use the same TLS/SSL certificate (because they will all identify themselves by the same host name), so only one certificate needs to be created.

All of the steps in this section should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

Generate a private key and certificate signing request

The private key and certificate signing request can be generated using either the `openssl` command or the Java `keytool` command. The former creates keys and certificates in standard formats that have to be imported to a Java keystore for use by Tomcat; the latter creates a Java keystore from which keys and certificates have to be exported for use by non-Java applications. There really isn't any technical reason to choose one tool over the other; use whichever one you're most comfortable with. In this document we will use `openssl` and then import to a Java keystore in the next step. Run the commands

```
casdev-master# cd /etc/pki/tls/private
casdev-master# openssl req -nodes -newkey rsa:2048 -sha256 -keyout ca
sdev.key -out casdev.csr
Generating a 2048 bit RSA private key
.........+++
............+++
writing new private key to 'casdev.key'
-----
You are about to be asked to enter information that will be incorpora
ted
into your certificate request.
What you are about to enter is what is called a Distinguished Name o
r a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []: New York
Locality Name (eg, city) [Default City]: New York
Organization Name (eg, company) [Default Company Ltd]: The New School
Organizational Unit Name (eg, section) []: IT
Common Name (eg, your name or your server's hostname) []: casdev.news
chool.edu
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
casdev-master#
```

to generate a private key and certificate signing request. (Replace the contents of the Distinguished Name fields with values appropriate for your organization.) Submit the certificate signing request (`casdev.csr`) to your certificate authority to obtain a certificate. When the certificate comes back from the certificate authority, copy it and any intermediate certificate(s) into `/etc/pki/tls/certs`, saving them as `casdev.crt`, `casdev-intermediate.crt`, etc. If your certificate authority offers multiple certificate formats, opt for the PEM format, which looks like:

```
-----BEGIN CERTIFICATE-----
AQEFAAOCAQ8AMIIBCgKCAQEAtGCKiysqhQF4/AA5Pvi7EIIRqbtVx/IF0CAFK8lv
6uDJDHjd7bSNhhzYJxUNCdN0DacYT5wI/s4n3mLEXQrIt0KsUdPD+s7qP9Lw05hI
WaG7KhP6RZ+UtWSvHwIZJUHvlJvh2GlARw/XwV3iHG3mxfl5nCLNihAR9S1r2qEY
...several more lines of base64-encoded data...
-----END CERTIFICATE----
```

Import the certificate into a Java keystore

To import a certificate into a Java keystore, the first step is to combine the certificate chain (the host certificate and any intermediate certificates) into a single file. If you are using a self-signed certificate or a certificate authority whose root certificate is not distributed with RHEL 7, include the root certificate as well. The certificates must be placed in certificate chain order from “lowest” to “highest” as shown:

```
casdev-master# cd /etc/pki/tls/certs
casdev-master# cat casdev.crt casdev-intermediate.crt [root.crt] > /o
pt/tomcat/casdev-all.crt
```

Next, create a PKCS#12-format file from the combined certificates and the private key file:

```
casdev-master# cd /opt/tomcat
casdev-master# openssl pkcs12 -export -inkey /etc/pki/tls/private/cas
dev.key -in casdev-all.crt -name tomcat -out casdev.p12
Enter Export Password: changeit
Verifying - Enter Export Password: changeit
casdev-master#
```

Be sure to enter a non-blank password, or the import command (next) will fail. Next, import the PKCS#12-format file to a Java keystore:

```
casdev-master# keytool -importkeystore -srckeystore casdev.p12 -srcst
oretype pkcs12 -destkeystore keystore.jks
Enter destination keystore password: changeit
Re-enter new password: changeit
Enter source keystore password: changeit
Entry for alias tomcat successfully imported.
Import command completed:  1 entries successfully imported, 0 entrie
s failed or cancelled
casdev-master#
```

Warning: The default keystore password used by Tomcat is `changeit`, hence its use in the examples above (and further below). Obviously, something other than this default value should be used in a production Tomcat deployment.

Finally, delete the intermediate files and set the appropriate permissions on the keystore file:

```
casdev-master# rm -f casdev-all.crt casdev.p12
casdev-master# chown root.tomcat keystore.jks
casdev-master# chmod 640 keystore.jks
```

Configure Tomcat server settings

Edit the file `/opt/tomcat/latest/conf/server.xml` and make the changes described below to disable unneeded network connectors and to enable and properly configure the TLS/SSL connector.

Disable the `SHUTDOWN` port

Locate the `Server` XML tag (around line 22), which should look something like this:

```
<Server port="8005" shutdown="SHUTDOWN">
```

We will be using `systemd` to manage Tomcat (see Configure `systemd` to start Tomcat (page 56)), so the `SHUTDOWN` port is not needed. Change the port number to `-1` to disable it, as shown below:

```
<Server port="-1" shutdown="SHUTDOWN">
```

Disable the HTTP connector

Locate the first definition of an HTTP connector on port 8080 (around line 69), which should look something like this:

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

As discussed previously, all CAS communications should occur over a secure (TLS) channel, so this connector is not needed. Comment it out by inserting `<!--` and `-->` around it, like this:

```
<!--
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
-->
```

Enable and configure the HTTPS connector

Locate the first definition of a TLS/SSL connector on port 8443 (around line 88). As shipped with Tomcat, it will be commented out and look something like this:

```
<!--
 <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioP
rotocol"
            maxThreads="150" SSLEnabled="true">
     <SSLHostConfig>
         <Certificate certificateKeystoreFile="conf/localhost-rsa.jk
s"
                      type="RSA" />
     </SSLHostConfig>
 </Connector>
 -->
```

Remove the comment lines (`<!--` and `-->`) and change the connector definition to look like this:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
    sslImplementationName="org.apache.tomcat.util.net.openssl.OpenSSLImplementation"
    SSLEnabled="true" connectionTimeout="20000" maxThreads="150">
    <SSLHostConfig
        ciphers="ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS"
        honorCipherOrder="true" protocols="all,-SSLv2Hello,-SSLv2,-SSLv3"
        disableSessionTickets="true">
        <Certificate
            certificateKeystoreFile="/opt/tomcat/keystore.jks"
            certificateKeystorePassword="changeit"
            type="RSA" />
    </SSLHostConfig>
    <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
</Connector>
```

To obtain the most up-to-date list of ciphers for the `ciphers` attribute, use the Mozilla SSL Configuration Generator and select “Apache” and “Intermediate.” Then copy and paste the value of the `SSLCipherSuite` parameter.

> ⚠ **Important:** The value of the `certificateKeystorePassword` attribute should be the same password you entered for the keystore file in Import the certificate into a Java keystore (page 44), above.

> ⓘ **Note:** The configuration above uses TCP port 8443 for the HTTPS (TLS/SSL) port. This is the conventional port used by Tomcat and CAS, but is not a

requirement. It's also possible, for example, to use the well-known HTTPS port (TCP 443) or any other port, simply by changing the value of the port attribute on the connector definition.

Disable the AJP connector

Locate the definition of the AJP (Apache JServ Protocol) connector on port 8009 (around line 115), which should look something like this:

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

AJP is mostly used when front-ending Tomcat with Apache HTTPD. We're not doing that, so this connector is not needed. Comment it out by inserting `<!--` and `-->` around it, like this:

```
<!-- <Connector port="8009" protocol="AJP/1.3" redirectPort="8443"
/> -->
```

References

- Tomcat Configuration Reference: The HTTP Connector
- Mozilla Wiki: Security/Server Side TLS
- Digital Ocean: OpenSSL Essentials
- Acmetek: Java Keytool Commands

Configure asynchronous request support

CAS 5 requires the servlet container to support asynchronous requests. To enable asynchronous request support in Tomcat, edit the file `/opt/tomcat/latest/conf/web.xml`, locate the definition of the default web application servlet (around line 103), and add the `<async-supported>` directive:

```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</servl
et-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>listings</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
    <async-supported>true</async-supported>
</servlet>
```

Then locate the definition of the JSP compiler and execution servlet (around line 251) and make the same addition:

```
<servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-clas
s>
    <init-param>
        <param-name>fork</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>xpoweredBy</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
    <async-supported>true</async-supported>
</servlet>
```

The steps above should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

References

- CAS 5: Servlet Container Configuration

Configure `X-Forwarded-For` header processing

When Tomcat is installed behind a load balancer as it will be in our environment, incoming network connections will have a source address of the load balancer's internal interface rather than the address of the client system on the other side of the load balancer. This will have a negative impact on Tomcat (as well as CAS) logging, since the logs will not be able to identify individual systems by their network addresses. It will also prevent certain CAS features, such as adaptive authentication based on client address geolocation, from working correctly.

To correct this situation, most load balancers can be configured to insert an `X-Forwarded-For` HTTP header into the data stream to identify the address of the connecting client system, and Tomcat can be configured to look for this header and substitute the address provided for the source address attached to the connection from the load balancer.

To configure Tomcat to process `X-Forwarded-For` HTTP headers, edit the file `/opt/tomcat/latest/conf/server.xml` again and locate the definition of the `AccessLogValve` (around line 160, after inserting the changes in Configure TLS/SSL settings (page 42)) and

1. Insert a `RemoteIpValve` definition above it.
2. Add a `requestAttributesEnabled` attribute to the `AccessLogValve` definition.

When finished, things should look something like this:

```
<!-- RemoteIp valve, process X-Forwarded-For headers
     Documentation at: /docs/config/valve.html -->
<Valve className="org.apache.catalina.valves.RemoteIpValve"
       internalProxies="192\.168\.1\.10" />

<!-- Access log processes all example.
     Documentation at: /docs/config/valve.html
     Note: The pattern used is equivalent to using pattern="common"
-->
<Valve className="org.apache.catalina.valves.AccessLogValve" director
y="logs"
       prefix="localhost_access_log" suffix=".txt"
       requestAttributesEnabled="true"
       pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

The value of the `internalProxies` attribute on the `RemoteIpValve` declaration should be a regular expression that matches the IP address(es) of the internal interface(s) of the load balancer(s). Since '.' is a special character in regular expressions, it should be escaped with a backslash. To represent multiple IP addresses, separate them with '|' symbols (for example, `192\.168\.1\.10|192\.168\.1\.20`).

The steps above should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

References

- Tomcat Configuration Reference: The Valve Component

Tune resource caching settings

To improve performance, Tomcat is configured by default to cache static resources. However, the size of the cache is too small to work effectively with the CAS application. To tune Tomcat's cache settings, edit the file `/opt/tomcat/latest/conf/context.xml`, locate the definition of the default context (around line 19), and add a `<Resources>` directive at the bottom:

```
<Context>
    <!-- Default set of monitored resources. If one of these change
s, the    -->
    <!-- web application will be reloade
d.                                     -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat
restarts -->
    <!--
    <Manager pathname="" />
    -->

    <!-- Enable caching, increase the cache size (10240 default), inc
rease   -->
    <!-- the ttl (5s defaul
t)                                                  -->
    <Resources cachingAllowed="true" cacheMaxSize="40960" cacheTtl="6
0000" />
</Context>
```

The step above should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

Configure asynchronous logging support

CAS 5's logging subsystem automatically inserts itself into the runtime application context at startup time, and is designed to clean up the logging context when Tomcat shuts down. Unfortunately, the default Tomcat JarScanner configuration skips over JAR files named `log4j*.jar`, which prevents this feature from working.

To correct this problem, edit the file `/opt/tomcat/latest/conf/catalina.properties` and locate the lines defining the `jarsToSkip` property (around lines 108-134), and then the specific line of that definition that includes `log4j*.jar` (around line 128):

```
tomcat.util.scan.StandardJarScanFilter.jarsToSkip=\
...
jmx-tools.jar,jta*.jar,log4j*.jar,mail*.jar,slf4j*.jar,\
...
```

and remove `log4j*.jar` from that line:

```
tomcat.util.scan.StandardJarScanFilter.jarsToSkip=\
...
jmx-tools.jar,jta*.jar,mail*.jar,slf4j*.jar,\
...
```

The step above should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

References

- CAS 5: Servlet Container Configuration
- Tomcat Configuration Reference: The Jar Scanner Component

Open TLS/SSL port in the firewall

For client systems to be able to communicate with the CAS server, the TCP port that Tomcat's HTTPS connector was configured to use earlier (see Enable and configure the HTTPS connector (page 46)) must be opened in the operating system firewall. To do this, first create a `firewalld` service configuration file called `/etc/firewalld/services/tomcat-https.xml` with the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>Tomcat Secure HTTP (HTTPS)</short>
  <description>Tomcat typically implements TLS/SSL-secured HTTP (HTTP
S) on a different port than a regular web server does (often so that
both servers can co-exist on the same system).</description>
  <port protocol="tcp" port="8443"/>
</service>
```

to define the Tomcat HTTPS service. Then, run the commands

```
casdev-master# restorecon /etc/firewalld/services/tomcat-https.xml
casdev-master# chmod 640 /etc/firewalld/services/tomcat-https.xml
```

to assign the correct SELinux context and file permissions to the `tomcat-https.xml` file. Finally, run the commands

```
casdev-master# firewall-cmd --zone=public --add-service=tomcat-https
--permanent
success
casdev-master# firewall-cmd --reload
success
casdev-master#
```

to open the newly-defined service in the system firewall.

The steps above should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

Configure `systemd` to start Tomcat

RHEL 7 uses `systemd` (instead of `init`) to manage system resources. A *unit* is any resource that `systemd` knows how to operate on and manage.

The steps below should be performed on the master build server (***casdev-master***); the results will be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) later.

Define Tomcat as a service unit

Create the file `/etc/systemd/system/tomcat.service` with the following contents to define Tomcat as a service unit to `systemd`:

```
[Unit]
Description=Apache Tomcat Web Application Container
After=network.target

[Service]
Type=forking
PIDFile=/var/run/tomcat.pid
UMask=0007

# Tomcat variables
Environment='JAVA_HOME=/usr/lib/jvm/java-openjdk'
Environment='CATALINA_PID=/var/run/tomcat.pid'
Environment='CATALINA_HOME=/opt/tomcat/latest'
Environment='CATALINA_BASE=/opt/tomcat/latest'
Environment='CATALINA_OPTS=-Xms512M -Xmx2048M -XX:+UseParallelGC -ser
ver'

# Needed to make use of Tomcat Native Library
Environment='LD_LIBRARY_PATH=/opt/tomcat/latest/lib'

ExecStart=/opt/tomcat/latest/bin/jsvc \
            -Dcatalina.home=${CATALINA_HOME} \
            -Dcatalina.base=${CATALINA_BASE} \
            -Djava.awt.headless=true \
            -Djava.util.logging.manager=org.apache.juli.ClassLoaderLo
gManager \
            -Djava.util.logging.config.file=${CATALINA_BASE}/conf/log
ging.properties \
            -cp ${CATALINA_HOME}/bin/commons-daemon.jar:${CATALINA_HO
ME}/bin/bootstrap.jar:${CATALINA_HOME}/bin/tomcat-juli.jar \
            -pidfile ${CATALINA_PID} \
            -java-home ${JAVA_HOME} \
            -user tomcat \
            $CATALINA_OPTS \
            org.apache.catalina.startup.Bootstrap

ExecStop=/opt/tomcat/latest/bin/jsvc \
            -pidfile ${CATALINA_PID} \
            -stop \
            org.apache.catalina.startup.Bootstrap

[Install]
WantedBy=multi-user.target
```

⚠ **Important:** Although a 2 GB maximum size for the Java heap is adequate for CAS development and testing, production operation requires more resources. When installing on the production servers, change `-Xmx2048M` to `-Xmx4096M` in the “Tomcat variables” section.

Enable the Tomcat service unit

Run the commands

```
casdev-master# restorecon /etc/systemd/system/tomcat.service
casdev-master# chmod 644 /etc/systemd/system/tomcat.service
```

to assign the correct SELinux context and file permissions to the `tomcat.service` file, and run the command

```
casdev-master# systemctl enable tomcat.service
```

to enable the Tomcat service in `systemd`. This will cause `systemd` to start Tomcat at system boot time. Additionally, the following commands may now be used to manually start, stop, restart, and check the status of the Tomcat service:

```
casdev-master# systemctl start tomcat
casdev-master# systemctl stop tomcat
casdev-master# systemctl restart tomcat
casdev-master# systemctl status tomcat
```

Test the Tomcat installation

Before distributing the Tomcat installation to the CAS servers, it should be tested on the master build server (***casdev-master***) to ensure that everything is working properly.

Modify the ROOT web application to identify the server and client

Before starting the testing, edit the file `/opt/tomcat/latest/webapps/ROOT/index.jsp` and find the “congrats” section (around line 51):

```
<div id="congrats" class="curved container">
    <h2>If you're seeing this, you've successfully installed Tomcat.
Congratulations!</h2>
</div>
```

Insert the text shown below to display the host name, IP address, and port number of the Tomcat server:

```
<div id="congrats" class="curved container">
    <h2>If you're seeing this, you've successfully installed Tomcat.
Congratulations!</h2>
    <p>Server:
       <%= request.getLocalName() %> /
       <%= request.getLocalAddr() %> /
       <%= request.getLocalPort() %></p>
    <p>Client:
       <%= request.getRemoteHost() %> /
       <%= request.getRemoteAddr() %> /
       <%= request.getRemotePort() %></p>
</div>
```

This will be helpful later when testing the CAS servers through the load balancer, to ensure that requests are being distributed across the server pool, and also to ensure that `X-Forwarded-For` header processing is working correctly.

Start Tomcat

Start the Tomcat server by running the command

```
casdev-master# systemctl start tomcat
```

Review the contents of the log file (`/var/log/tomcat/catalina.yyyy-mm-dd.out`) for errors. All log messages in a successful start should be at log level `INFO`. If any messages are at log level `WARNING` or `SEVERE`, then something is wrong and needs to be corrected.

Note: The Tomcat SSL connector configuration created in Configure TLS/SSL settings (page 42) includes a setting for the `disableSessionTickets` property (setting it to `true`). This is necessary to avoid a bug in Tomcat's JSSE with OpenSSL implementation (BugID 59811). However, including the property will cause this warning message to appear:

```
DD-MMM-YYYY HH:MM:SS.sss WARNING [main]
org.apache.tomcat.util.net.SSLHostConfig.setConfigType The property
[disableSessionTickets] was set on the SSLHostConfig named
[_default_] and is for connectors of type [OPENSSL] but the
SSLHostConfig is being used with a connector of type [EITHER]
```

This warning message is being printed erroneously (as configured, the JSSE connector *is* using OpenSSL) and should be ignored.

The last line of the log file in a successful start should look like this:

```
DD-MMM-YYYY HH:MM:SS.sss INFO [main] org.apache.catalina.startup.Cata
lina.start Server startup in N ms
```

Check that the Tomcat Native Library was correctly installed

After reviewing the log file in general, look specifically for messages that the Tomcat Native Library was successfully loaded and that it's using the OpenSSL library:

```
DD-MMM-YYYY HH:MM:SS.sss INFO [main] org.apache.catalina.core.AprLife
cycleListener.lifecycleEvent Loaded APR based Apache Tomcat Native li
brary [1.2.12] using APR version [1.6.2].
DD-MMM-YYYY HH:MM:SS.sss INFO [main] org.apache.catalina.core.AprLife
cycleListener.initializeSSL OpenSSL successfully initialized [OpenSS
L 1.1.0f  25 May 2017]
```

If instead a message like this appears:

```
DD-MMM-YYYY HH:MM:SS.sss INFO [main] org.apache.catalina.core.AprLife
cycleListener.lifecycleEvent The APR based Apache Tomcat Native libra
ry which allows optimal performance in production environments was no
t found on the java.library.path: /usr/java/packages/lib/amd64:/usr/l
ib64:/lib64:/lib:/usr/lib
```

then the Native Library was not installed correctly, and the installation needs to be fixed.

Access the ROOT web application

Open up a web browser and enter the URL of Tomcat's TLS port on the master build server:

```
https://casdev-master.newschool.edu:8443
```

Expect the browser to complain about the TLS/SSL certificate because the host name of the server does not match the name in the certificate. Click through the prompts to visit the site anyway, and you should see something like Figure 3, below:

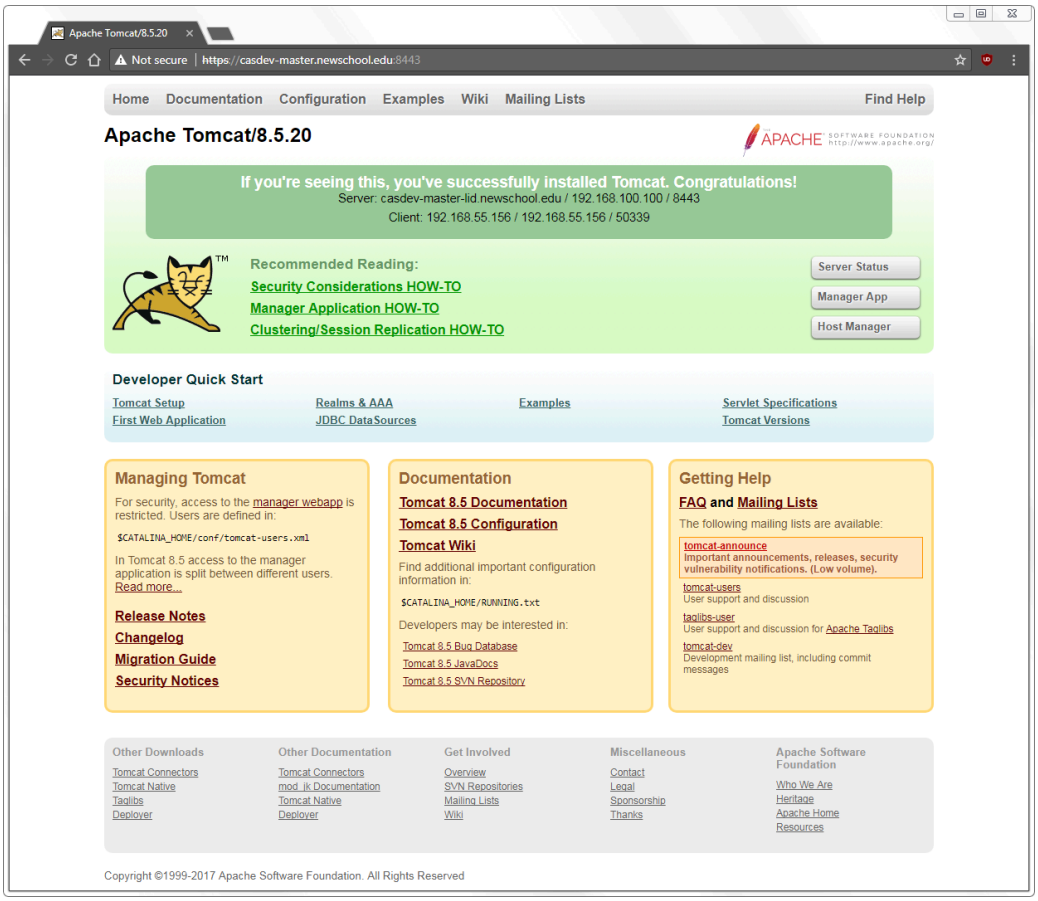

Figure 3. The ROOT web application page

Notice the host name, IP address, and port number of the server and client displayed in the top (dark green) section; this will be important when testing through the load balancer later.

Using the features of your web browser, examine the details of the TLS certificate. (In Google Chrome, this is done by typing `Ctrl+Shift+I` to bring up the Developer Tools window, selecting the “Security” tab, and clicking “View certificate.”) Check that the certificate is the one created and installed in Configure TLS/SSL settings (page 42) and that the certificate chain is valid from the root to the certificate.

> ℹ **Note:** The gray buttons for “Server Status,” “Manager App,” and “Host Manager,” as well as most of the links in the blue and yellow areas of the page, will not work because they point to pages or example web applications that were deleted in the server hardening step.

Distribute the Tomcat installation to the CAS servers

Once the Tomcat server has been successfully built, configured, and tested on the master build server (***casdev-master***) as described in the previous sections, the installation can be copied to the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***). It is not necessary (or desirable) to install Tomcat on ***casdev-casapp*** or ***casdev-samlsp***.

Create a distribution `tar` file

To distribute all relevant files in the Tomcat installation to the CAS servers, we will assemble them all into a single `tar` archive that can be copied to each server and extracted. First, run the commands

```
casdev-master# systemctl stop tomcat
casdev-master# cd /opt/tomcat/latest
casdev-master# rm -rf logs/* work/*
```

to shut down the Tomcat server and remove files that don't need to be included in the archive. Then run the commands

```
casdev-master# cd /
casdev-master# tar czf /tmp/tomcat-files.tgz etc/tomcat opt/apr opt/o
penssl opt/tomcat var/cache/tomcat var/lib/tomcat var/cache/tomcat et
c/firewalld/services/tomcat-https.xml etc/systemd/system/tomcat.servi
ce
```

to create the `tar` archive in `/tmp/tomcat-files.tgz`.

Create an installation shell script

In addition to extracting the `tar` archive on each CAS server, commands must be executed to create the `tomcat` user and group, reset the correct user and group ownership of the installation, open firewall ports, and start the Tomcat service. To make it easier to run these commands on each server, they can be collected into a shell script (called, for example, `/opt/scripts/tomcat-install.sh`) like this:

```
#!/bin/sh

echo "--- Installing on `hostname`"

if [ -f /tmp/tomcat-files.tgz ]
then
    cd /
    tar xzf /tmp/tomcat-files.tgz

    groupadd -r tomcat
    useradd -r -d /opt/tomcat -g tomcat -s /sbin/nologin tomcat

    for dir in . conf webapps
    do
        cd /opt/tomcat/latest/$dir
        chown -R root.tomcat .
    done

    for dir in logs temp work
    do
        cd /opt/tomcat/latest/$dir
        chown -R tomcat.tomcat .
    done

    restorecon /etc/firewalld/services/tomcat-https.xml
    firewall-cmd --zone=public --add-service=tomcat-https --permanent
    firewall-cmd --reload

    restorecon /etc/systemd/system/tomcat.service
    systemctl enable tomcat.service
    systemctl start tomcat

    rm -f /tmp/tomcat-files.tgz /tmp/tomcat-install.sh
    echo "Installation complete."
else
    echo "Cannot find /tmp/tomcat-files.tgz; nothing installed."
fi

exit 0
```

This script will extract the contents of the tar archive to the right places and then run all the necessary commands to finish the installation and start Tomcat.

> ℹ **Note:** There is nothing special about the directory `/opt/scripts` (you can create it wherever you like, or not at all), but since we will be creating several

little “helper” shell scripts throughout this deployment process, it makes sense to collect them all in a common location on the master build server.

Copy files to each server and run the installation script

To complete the setup of each CAS server, the `tar` archive and installation shell script need to be copied to each server, and the installation script executed. This can be done manually, or with a shell loop as shown below:

```
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/tomcat-files.tgz casdev-${host}:/tmp/tomcat-files.tgz
> scp -p /opt/scripts/tomcat-install.sh casdev-${host}:/tmp/tomcat-in
stall.sh
> ssh casdev-${host} sh /tmp/tomcat-install.sh
> done
casdev-master#
```

Test Tomcat on each server

Open up a web browser and enter the URL of Tomcat’s TLS port on each CAS server:

```
https://casdev-srv01.newschool.edu:8443
https://casdev-srv02.newschool.edu:8443
https://casdev-srv03.newschool.edu:8443
```

As in the previous test, expect the browser to complain about the TLS/SSL certificate because the host name of the server does not match the name in the certificate. Verify that the Tomcat “success” page appears on each server, just as it did for the development server.

Configure the load balancers

Once all the CAS servers have Tomcat up and running, we can configure the load balancers to route requests to them.

> ℹ **Note:** The configuration steps in this section are for F5 BIG-IP Local Traffic Manager (LTM) load balancers; other load balancers should provide similar capabilities.

> **Correcting our biggest mistake**
>
> Our current CAS server listens for connections on TCP port 8447 instead of the more common 8443 or 443. The history of this decision isn't important, but experience has taught us that it was a Really Big Mistake. Many organizations limit outbound Internet access from their local area networks to small(ish) lists of well-known ports or services, and TCP port 8447 is almost never on those lists.
>
> If access to port 8447 is blocked, then users have no way to authenticate to CAS-enabled New School applications. This has been a problem for some of our users when using the EDUROAM roaming access service at other universities, and also for some of our students and part-time faculty who work at financial firms or other security-conscious organizations in New York City. Port 8443 (the "usual" port used by CAS) would have been a somewhat better choice, but not a perfect one, because there are organizations that block outbound access to that port as well.
>
> To eliminate this problem, and ensure that our users can authenticate regardless of what network they're using, our new CAS environment will be configured to listen for connections on TCP port 443, the standard HTTPS port.

Define the CAS server nodes

Define each of the CAS server nodes so that they can be referenced elsewhere in the configuration.

```
ltm node /Common/casdev-srv01 {
    address 192.168.100.101
    description "CAS Development Server 01"
}
ltm node /Common/casdev-srv02 {
    address 192.168.100.102
    description "CAS Development Server 02"
}
ltm node /Common/casdev-srv03 {
    address 192.168.100.103
    description "CAS Development Server 03"
}
```

Define the CAS server pool

Group the CAS server nodes into a server pool. Pool members can be assigned to connections via round-robin or any other reasonable method that achieves the organization's desired results.

```
ltm pool /Common/casdev_pool {
    description "CAS Development 8443 Pool"
    members {
        /Common/casdev-srv01:8443 {
            address 192.168.100.101
        }
        /Common/casdev-srv02:8443 {
            address 192.168.100.102
        }
        /Common/casdev-srv03:8443 {
            address 192.168.100.103
        }
    }
    monitor /Common/https_8443
}
```

Define a monitor for the pool to keep track of which servers are up and responding to requests. The `https_8443` monitor is based on F5's standard `https` monitor; it connects to each pool server via HTTPS on port 8443 every 5 seconds and issues a `GET /` HTTP request.

```
ltm monitor https /Common/https_8443 {
    cipherlist DEFAULT:+SHA:+3DES:+kEDH
    compatibility enabled
    defaults-from /Common/https
    description "HTTPS Port 8443 Port Monitor"
    destination *:8443
    interval 5
    ip-dscp 0
    recv none
    recv-disable none
    send "GET /\\r\\n"
    time-until-up 0
    timeout 16
}
```

This is a basic monitor that just checks whether Tomcat and the server it's running on are responding. Later, after building and deploying the CAS server application, we will configure a more specific monitor (see Define a CAS-specific service monitor on the load balancers (page 104)).

Define a client SSL profile

Define a client SSL profile to enable the F5 to accept and terminate client requests using TLS/SSL. This profile specifies the TLS/SSL certificate that will be used for the connection.

```
ltm profile client-ssl /Common/casdev_clientssl {
    app-service none
    cert /Common/casdev.crt
    cert-key-chain {
        casdev_ThawteSHA256IntermediateCA_Use_for_SHA256-NoSHA1crossr
oot {
            cert /Common/casdev.crt
            chain /Common/ThawteSHA256IntermediateCA_Use_for_SHA256-N
oSHA1crossroot.crt
            key /Common/casdev.key
        }
    }
    chain /Common/ThawteSHA256IntermediateCA_Use_for_SHA256-NoSHA1cro
ssroot.crt
    defaults-from /Common/nsu_clientssl
    inherit-certkeychain false
    key /Common/casdev.key
    passphrase none
}
```

The TLS/SSL private key and certificate files referenced in this profile are the same ones that were created for the Tomcat server in Configure TLS/SSL settings (page 42); they should be installed on the load balancer for use by the profile.

Define a server profile

Define a server SSL profile to direct the F5 to access the Tomcat server pool using HTTPS instead of HTTP.

```
ltm profile server-ssl /Common/casdev_serverssl {
    app-service none
    defaults-from /Common/serverssl
}
```

Define a persistence profile

Although the basic CAS login sequence is stateless, there are some features of the server that implement flows whose steps must all be performed on the same server to preserve state. To achieve this, define a persistence profile with a timeout equal to the `server.session.timeout` property of the CAS server (5 minutes by default).

```
ltm persistence source-addr /Common/casdev_persistence_profile {
    app-service none
    defaults-from /Common/cookie
    expiration 5:0
}
```

The above profile uses a session cookie based persistence profile in which the F5 sets a cookie in the user's browser. This will ensure that all connection requests from the browser session where the cookie is set are directed to the same pool member for the duration of the timeout period. Another alternative would be to use a source address based persistence profile, which would ensure that all connection requests from a particular IP address are directed to the same pool member. The cookie based approach is preferred as it is more granular, resulting in better load balancing performance.

Enable the insertion of `X-Forwarded-For` headers

In Configure X-Forwarded-For header processing (page 51), we configured Tomcat to process `X-Forwarded-For` HTTP headers inserted by a load balancer. Define an HTTP profile on the F5 to enable the insertion of those headers.

```
ltm profile http /Common/http-casdev-profile {
    app-service none
    defaults-from /Common/http
    enforcement {
        unknown-method allow
    }
    insert-xforwarded-for enabled
    proxy-type reverse
}
```

Define the virtual interface

Define the virtual interface that will listen on the user-facing side of the load balancer. As discussed at the beginning of this section, we want our CAS service to be available on TCP port 443, the standard HTTPS port. Configure the virtual interface to listen for connections on TCP port 443 and redirect them to TCP port 8443 on one of the pool servers. Set the interface to use the persistence profile defined above.

```
ltm virtual /Common/casdev_https_vs {
    description "CAS Development https VIP"
    destination /Common/192.168.200.10:443
    ip-protocol tcp
    mask 255.255.255.255
    persist {
        /Common/casdev_persistence_profile {
            default yes
        }
    }
    pool /Common/casdev_pool
    profiles {
        /Common/casdev_clientssl {
            context clientside
        }
        /Common/casdev_serverssl {
            context serverside
        }
        /Common/http-casdev-profile { }
        /Common/tcp { }
    source 0.0.0.0/0
    source-address-translation {
        type automap
    }
    translate-address enabled
    translate-port enabled
}
```

As discussed previously, CAS communications should always take place over a secure channel. Configure the virtual interface to redirect HTTP connections to HTTPS.

```
ltm virtual /Common/casdev_http_vs {
    description "CAS Development http VIP"
    destination /Common/192.168.200.10:80
    ip-protocol tcp
    mask 255.255.255.255
    profiles {
        /Common/http { }
        /Common/tcp { }
    }
    rules {
        /Common/http_to_https_irule
    }
    source 0.0.0.0/0
    translate-address enabled
    translate-port enabled
}
```

Test the servers through the load balancer

Once the F5 has been configured, repeat the testing performed earlier using the virtual address assigned to the load balancer's virtual interface:

```
https://casdev.newschool.edu
```

Since this host name corresponds to the host name in the TLS certificate installed on the CAS servers, check to ensure that no browser warnings about the certificate appear, and that the certificate chain is valid all the way back up to the root certificate authority.

Perform tests from multiple client systems with different addresses to ensure that the round-robin (or other selected method) of distributing requests across the pool is working properly.

Confirm that the IP address of the client system is displayed in the top (dark green) section of the page rather than the IP address of the load balancer. This will indicate that the `X-Forwarded-For` header processing has been properly configured.

Perform a TLS/SSL check on the servers (optional)

If the load balancer's virtual interface is accessible from the Internet, use the Qualys® SSL Labs SSL Server Test to check that TLS/SSL is correctly configured and that the server receives an overall 'A' rating. If any other rating is received, check the test report for errors and correct them.

If the load balancer's virtual interface is not accessible from the Internet, use the `testssl.sh` command line tool instead. This tool performs a similar battery of tests; the principal difference is that it doesn't assign a letter grade to the overall results.

Install HTTPD and PHP on the client servers

Summary: A CAS-enabled Apache HTTPD server and a SAML2-enabled Apache HTTPD server will be used as clients to test the CAS server. PHP will be used to help examine user attribute data passed back to the clients from the server.

To test the operation of our CAS server environment, we need a client application that will prompt for a user name and password (and perhaps other authentication factors) and then contact the CAS server to authenticate that user, and request/accept attribute information describing the user. Since the development environment will support both the CAS and SAML2 protocols for this purpose, we need two client applications—one for each protocol.

There are a variety of open-source and commercial libraries and toolkits that can be used to build CAS or SAML2 support into client applications, but building client applications from scratch is beyond the scope of this project. Therefore, we will instead use the Apache HTTPD server as our client, with CAS support enabled by the Apereo `mod_auth_cas` plug-in, and SAML2 support enabled by the Shibboleth Consortium's Service Provider distribution. We will install the clients on two separate servers, ***casdev-casapp.newschool.edu*** and ***casdev-samlsp.newschool.edu***.

This section describes the Apache HTTPD and PHP (for processing attributes) installation steps common to both servers; the protocol-specific configuration of each server will be described in later sections.

References

- Apereo mod_auth_cas
- Shibboleth Service Provider
- CAS 5: CAS Clients
- Wikipedia: Libraries and toolkits to develop SAML actors and SAML-eanbled services

Install software packages

As discussed in the introduction to this section (page 74), we need to install Apache HTTPD and PHP on the servers. We also need to install the `mod_ssl` plugin for Apache, which enables TLS/SSL support. Run the commands

```
# yum -y install httpd
# yum -y install mod_ssl
# yum -y install php
```

on ***casdev-casapp*** and ***casdev-samlsp*** to install these packages. It is not necessary (or desirable) to install HTTPD and PHP on ***casdev-srv01***, ***casdev-srv02***, or ***casdev-srv03***.

Configure TLS/SSL settings

Apache HTTPD's TLS/SSL settings must be configured and a TLS/SSL certificate must be provided to ensure that all communications with the CAS server occur over a secure channel.

> ℹ **Note:** The steps below are shown for ***casdev-casapp***; they should also be performed on ***casdev-samlsp*** with host names substituted as appropriate.

Generate private keys and certificate signing requests

The private key and certificate signing request are generated using the `openssl` command. Run the commands

```
casdev-casapp# cd /etc/pki/tls/private
casdev-casapp# openssl req -nodes -newkey rsa:2048 -sha256 -keyout ca
sapp.key -out casapp.csr
Generating a 2048 bit RSA private key
.........+++
............+++
writing new private key to 'casapp.key'
-----
You are about to be asked to enter information that will be incorpora
ted
into your certificate request.
What you are about to enter is what is called a Distinguished Name o
r a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []: New York
Locality Name (eg, city) [Default City]: New York
Organization Name (eg, company) [Default Company Ltd]: The New School
Organizational Unit Name (eg, section) []: IT
Common Name (eg, your name or your server's hostname) []: casdev-casa
pp.newschool.edu
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
casdev-casapp#
```

to generate a private key and certificate signing request. (Replace the contents of the Distinguished Name fields with values appropriate for your organization.) Submit the certificate signing request (`casapp.csr`) to your certificate authority to obtain a certificate.

When the certificate comes back from the certificate authority, copy it and any intermediate certificate(s) into `/etc/pki/tls/certs`, saving them as `casapp.crt`, `casapp-intermediate.crt`, etc. If your certificate authority offers multiple certificate formats, opt for the PEM format, which looks like:

```
-----BEGIN CERTIFICATE-----
AQEFAAOCAQ8AMIIBCgKCAQEAtGCKiysqhQF4/AA5Pvi7EIIRqbtVx/IF0CAFK8lv
6uDJDHjd7bSNhhzYJxUNCdN0DacYT5wI/s4n3mLEXQrIt0KsUdPD+s7qP9Lw05hI
WaG7KhP6RZ+UtWSvHwIZJUHvlJvh2GlARw/XwV3iHG3mxfl5nCLNihAR9S1r2qEY
...several more lines of base64-encoded data...
-----END CERTIFICATE----
```

Configure HTTPD settings

Edit the file `/etc/httpd/conf/httpd.conf` and locate the `ServerName` directive (around line 95), which should look something like this:

```
#ServerName www.example.com:80
```

Uncomment the line and replace `www.example.com:80` with `casdev-casapp.newschool.edu`, and then add a `UseCanonicalName` directive on the next line, like this:

```
ServerName casdev-casapp.newschool.edu
UseCanonicalName on
```

Then, at the bottom of the file, add the following lines:

```
<VirtualHost *:80>
    Redirect permanent / https://casdev-casapp.newschool.edu/
</VirtualHost>
```

to automatically redirect any HTTP connections (port 80) to HTTPS connections (port 443), which will help ensure that all communications occur over a secure communications channel.

Configure TLS/SSL settings

Edit the file `/etc/httpd/conf.d/ssl.conf` and locate the `SSLProtocol` directive (around line 75) and the `SSLCipherSuite` directive (around line 80), the two of which should look something like this:

```
SSLProtocol all -SSLv2
SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5:!SEED:!IDEA
```

Change the values of these two directives, and add an `SSLHonorCipherOrder` directive, so that it all looks like this:

```
SSLProtocol all -SSLv2 -SSLv3
SSLCipherSuite ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1
305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-E
CDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GC
M-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RS
A-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDH
E-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECD
HE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AE
S256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DE
S-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:A
ES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
SSLHonorCipherOrder on
```

To obtain the most up-to-date values for these three attributes, use the Mozilla SSL Configuration Generator and select “Apache” and “Intermediate.” Then copy and paste the values given by the generator into the configuration above.

Then locate the `SSLCertificateFile`, `SSLCertificateKeyFile`, and `SSLCertificateChainFile` directives (around lines 100-116) and set them to the names of the server certificate file, private key file, and (if applicable) intermediate certificate file:

```
SSLCertificateFile /etc/pki/tls/certs/casapp.crt
SSLCertificateKeyFile /etc/pki/tls/private/casapp.key
SSLCertificateChainFile /etc/pki/tls/certs/casapp-intermediate.crt
```

Check the HTTPD Configuration

Check that the HTTPD configuration files edited above do not have any syntax errors by running the command

```
casdev-casapp# apachectl configtest
Syntax OK
casdev-casapp#
```

Configure PHP settings

Edit the file `/etc/php.ini` and locate the `date.timezone` setting (around line 878), which should look something like:

```
;date.timezone =
```

Uncomment the line and set the value to the local time zone:

```
date.timezone = America/New_York
```

The list of supported time zone names is available from the PHP List of Supported Timezones .

Open HTTP/HTTPS ports in the firewall

To communicate with client systems, HTTPD needs to be able to communicate on TCP ports 80 (HTTP) and 443 (HTTPS). Run the commands

```
# firewall-cmd --zone=public --add-service=http --permanent
success
# firewall-cmd --zone=public --add-service=https --permanent
success
# firewall-cmd --reload
success
#
```

on ***casdev-casapp*** and ***casdev-samlsp*** to open these ports in the system firewall.

Configure `systemd` to start HTTPD

RHEL 7 uses `systemd` (instead of `init`) to manage system resources. Run the command

```
# systemctl enable httpd.service
```

on ***casdev-casapp*** and ***casdev-samlsp*** to enable the HTTPD service in `systemd`. This will cause `systemd` to start HTTPD at system boot time. Additionally, the following commands may now be used to manually start, stop, restart, and check the status of the HTTPD service:

```
# systemctl start httpd
# systemctl stop httpd
# systemctl restart httpd
# systemctl status httpd
```

Test the HTTPD installation

Note: The steps below are shown for ***casdev-casapp***; they should also be performed on ***casdev-samlsp*** with host names substituted as appropriate.

Create a basic web page

Create the file `/var/www/html/index.php` with the following contents to make a basic web page that displays some simple content:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/
3.3.7/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <h1>Hello, World!</h1>
        <p><big>The quick brown fox jumped over the lazy dogs.</bi
g></p>
        <?php phpinfo(); ?>
    </div>
  </body>
</html>
```

Note: Inclusion of the Bootstrap stylesheet is optional; it just makes the page a little more readable.

Start HTTPD

Start the HTTPD server by running the command

```
# systemctl start httpd
```

Review the contents of the log files in the `/var/log/httpd` directory for errors.

Access the server

Open up a web browser and enter the HTTP URL of the server:

```
http://casdev-casapp.newschool.edu
```

Check that the server redirects the browser to the HTTPS version of the URL (the browser address bar should now display `https://casdev-casapp.newschool.edu`), and that you see something like this:

Figure 4. The test example web page

Perform a TLS/SSL check on the servers (optional)

If ***casdev-casapp*** and ***casdev-samlsp*** are accessible from the Internet, use the Qualys® SSL Labs SSL Server Test to check that TLS/SSL is correctly configured and that the servers receive an overall ‘A’ rating. If any other rating is received, check the test report for errors and correct them.

If ***casdev-casapp*** and ***casdev-samlsp*** are not accessible from the Internet, use the `testssl.sh` command line tool instead. This tool performs a similar battery of tests; the principal difference is that it doesn't assign a letter grade to the overall results.

Building the CAS server

Summary: Now that the development environment has been set up, CAS server development can begin with building and configuring a (very) basic server.

The Apache Maven build automation tool is used to configure and build the CAS server (CAS 4.2 and later also support using Gradle). Maven keeps track of the hundreds of library and object code dependencies associated with the CAS server and the particular features we have chosen to include, downloads the necessary files (in the appropriate versions) from public code repositories to a local cache, and assembles everything into a deployable bundle.

The CAS development team recommends that a WAR overlay project be used to organize feature selections and user interface design. This approach allows us to "overlay" our customizations—enabling or disabling features, setting configuration options, modifying the look and feel, etc.—onto a pre-built "vanilla" web application server provided by the CAS project itself, without having to download or build those components that we aren't using or changing.

We only have to manage the files that contain our customizations; Maven will take care of everything else.

Create a work area

Because we will be working with more than one WAR overlay project (we will be creating separate ones later for the management application and the cloud configuration server), we'll create a top-level directory to keep them all in. Run the command

```
casdev-master# mkdir /opt/workspace
```

to create a top-level directory on the master build server (***casdev-master***).

> ℹ **Note:** The directory may be created anywhere on the system; it does not have to reside under `/opt`. Furthermore, super-user permissions are not needed to build and configure the server (although they will be needed to deploy it).

References

- CAS 5: WAR Overlay Installation
- Apache Maven: Overlays

Create a Maven WAR overlay project

We will use the Maven WAR overlay template provided by the CAS project as the starting point for our own project.

Clone the overlay template project

Use Git to clone the overlay template project from GitHub. Run the commands

```
casdev-master# cd /opt/workspace
casdev-master# git clone https://github.com/apereo/cas-overlay-templa
te.git
Cloning into 'cas-overlay-template'...
remote: Counting objects: 610, done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 610 (delta 7), reused 12 (delta 4), pack-reused 594
Receiving objects: 100% (610/610), 198.63 KiB | 0 bytes/s, done.
Resolving deltas: 100% (298/298), done.
casdev-master#
```

on the master build server (***casdev-master***). This will make a local copy of all files in the template project and store them in a directory called `cas-overlay-template`. It will also record the information needed for Git to keep the local copy of the files synchronized with the copy stored on GitHub, so that corrections and updates made by the project team can be incorporated into our project from time to time.

> ☑ **Tip:** As an alternative to using Git to clone a repository, GitHub allows the files in a repository to be downloaded in a Zip archive. However, this method does not include the metadata that Git needs to keep the local copy in sync with the master repository.

Switch to the right branch

The GitHub repository for the overlay template project contains multiple versions of the template; each version is stored as a separate branch of the project. The `master` branch usually points to the version of the template used for configuring and deploying the latest stable release of the CAS server; this is the branch that will initially be copied to disk by cloning the project. Run the commands

```
casdev-master# cd cas-overlay-template
casdev-master# grep '<cas.version>' pom.xml
        <cas.version>5.2.0</cas.version>
casdev-master#
```

to determine which version of the CAS server the `master` branch will build. In most circumstances (including this project), the `master` branch of the template is the one you want to use (skip ahead to the next section, Create a local branch (page 90)).

If the `master` version of the template isn't for the version of the CAS server you want to work with (for example, if you want to work with an older version, or experiment with the version currently under development), run the command

```
casdev-master# git branch -a
* master
  remotes/origin/4.1
  remotes/origin/4.2
  remotes/origin/5.0.x
  remotes/origin/5.1
  remotes/origin/5.2
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
casdev-master#
```

to obtain a list of available branches, and then run the `git checkout` command to switch to that branch. For example, to switch back to the `5.1` branch, run the command

```
casdev-master# git checkout 5.1
Branch 5.1 set up to track remote branch 5.1 from origin.
Switched to a new branch '5.1'
casdev-master#  grep '<cas.version>' pom.xml
        <cas.version>5.1.5</cas.version>
casdev-master#
```

to switch branches (it's not necessary to type the `remotes/origin/` part of the branch name). This will download additional/changed files from GitHub to the local disk. You can switch back to the current version of the template by checking out the `master` branch again:

```
casdev-master# git checkout master
Switched to branch 'master'
casdev-master#  grep '<cas.version>' pom.xml
        <cas.version>5.2.0</cas.version>
casdev-master#
```

Create a local branch

After you're on the right branch (for our project, you should be on the `master` branch), create a new branch local to your project, which will be used to track all of your changes and keep them separate from any changes made to the template by the CAS developers. This will make it easier in the future to merge upstream changes from the CAS project team into your local template without having to redo all your changes.

Choose a meaningful name for your branch, but not somthing likely to be duplicated by the CAS developers—for example, `newschool-casdev`. Run the commands

```
casdev-master# git checkout -b newschool-casdev
Switched to a new branch 'newschool-casdev'
casdev-master#
```

to create this new branch (replace `newschool-casdev` with the name of your branch).

Build the default server

The Maven WAR overlay template will, out-of-the-box without any configuration, build a very basic “default” CAS server. This server doesn’t do much, but it will let us verify that everything we’ve done up to this point is working correctly and give us a starting point for further configuration and customization. To build the default server, run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# ./mvnw clean package
Downloading https://repository.apache.org/content/repositories/releas
es/org/apache/maven/apache-maven/3.5.0/apache-maven-3.5.0-bin.zip
Unzipping /root/.m2/wrapper/dists/apache-maven-3.5.0-bin/766bhoj4b69i
19aqdd66g707g1/apache-maven-3.5.0-bin.zip to /root/.m2/wrapper/dists/
apache-maven-3.5.0-bin/766bhoj4b69i19aqdd66g707g1
Set executable permissions for: /root/.m2/wrapper/dists/apache-mave
n-3.5.0-bin/766bhoj4b69i19aqdd66g707g1/apache-maven-3.5.0/bin/mvn
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(several hundred more lines of diagnostic output... check for errors)
```

on the master build server (***casdev-master***). Since this is our first time running `mvnw`, several hundred lines of diagnostic output will be printed as the wrapper downloads and installs Maven (to a cache directory), and as Maven downloads all the various software components that CAS servers are built from—CAS modules, Java libraries, third-party packages, etc.—from public software repositories and stores them in its cache. Once all that preparatory work is finished, the CAS application itself will be built:

```
[INFO] Packaging webapp
[INFO] Assembling webapp [cas-overlay] in [/opt/workspace/cas-overla
y-template/target/cas]
[info] Copying manifest...
[INFO] Processing war project
[INFO] Processing overlay [ id org.apereo.cas:cas-server-webapp-tomca
t]
[INFO] Webapp assembled in [1086 msecs]
[INFO] Building war: /opt/workspace/cas-overlay-template/target/cas.w
ar
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 02:10 min
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 13M/46M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

The end result of a successful build will be a subdirectory called `target` that contains a `cas.war` file:

```
casdev-master# ls -asl target
total 77148
    0 drwxr-xr-x. 5 root root       61 Mmm dd hh:mm .
    4 drwxr-xr-x. 6 root root     4096 Mmm dd hh:mm ..
    0 drwxr-xr-x. 5 root root       45 Mmm dd hh:mm cas
89076 -rw-r--r--. 1 root root 91210098 Mmm dd hh:mm cas.war
    0 drwxr-xr-x. 2 root root       27 Mmm dd hh:mm maven-archiver
    0 drwxr-xr-x. 3 root root       17 Mmm dd hh:mm war
casdev-master#
```

(ignore the other things in the `target` directory for now).

Configure server properties

By default, CAS expects to find its configuration files in the operating system directory `/etc/cas`. Almost every aspect of CAS server configuration is controlled via settings stored in the `cas.properties` file located in the `/etc/cas/config` directory. The Maven WAR overlay template provides a "source" for this file (which makes it easy to manage with Git).

Configure server name information

There are three properties that provide naming information to the CAS server:

`cas.server.name`	The top-level URL (protocol, domain name, and port) of the web/application server running the CAS server.
`cas.server.prefix`	The URL of the CAS web application on the web/application server. This string gets prepended to the various CAS-specific URLs used by the server.
`cas.host.name`	The name of the CAS host to be appended to ticket IDs. This value is normally determined automatically, but can be explicitly set in cases where that value may be incorrect (e.g., when hosting CAS servers for multiple domains on the same host).

Edit the file `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the lines for `cas.server.name` and `cas.server.prefix` properties at the top of the file. Set `cas.server.name` to the correct value by replacing `cas.example.org` with the host name attached to the virtual address on the load balancer's virtual interface and removing the port part of the URL (since we're running on the standard SSL/TLS port). Then, rather than duplicating that information for `cas.server.prefix`, use variable substitution to incorporate the value of `cas.server.name`:

```
cas.server.name:                    https://casdev.newschool.edu
cas.server.prefix:                  ${cas.server.name}/cas
```

Since we will (eventually) have multiple servers generating tickets, we want to leave `cas.host.name` unset (the default). This will result in each ticket having a ticket ID that includes the host name of the server that actually created the ticket,

which will make it easier to debug ticket issues. If we were to set `cas.host.name`, all the tickets would have the same “host name” in their ticket IDs, and it would be impossible to tell which server actually created the ticket.

Configure ticket granting cookie encryption

The CAS server uses a ticket granting cookie in the browser to maintain login state during single sign-on sessions. A client can present this cookie to CAS in lieu of primary credentials and, provided it is valid, will be authenticated. The contents of the cookie should be encrypted to protect them, and when running in a multi-node environment, all of the nodes must use the same keys. Add the following lines to `etc/cas/config/cas.properties`:

```
cas.tgc.secure:                         true
cas.tgc.crypto.signing.key:
cas.tgc.crypto.encryption.key:
```

Now visit the JSON Web Key Generator and click on the “Shared Secret” tab. Enter `512` into the “Key Size” field, select `HS256` from the “Algorithm” drop-down, and click the “New Key” button. Copy the value of the `k` parameter from the “Key” dialog box and enter it as the value for the `cas.tgc.crypto.signing.key` property.

Then enter `256` into the “Key Size” field, select `HS256` from the “Algorithm” drop-down, and click “New Key” again, and enter that value for the `cas.tgc.crypto.encryption.key` property. When finished, you should have something like this:

```
cas.tgc.secure:                         true
cas.tgc.crypto.signing.key:             bMpP_eHgIsL1kz_cnxEqYo9Bb384V70
eZIvWctQ5V6xTO4P6wsQjFlglD9OSQNlFdb0mT2Q1E3qXdo05_tzrjQ
cas.tgc.crypto.encryption.key:          r88iOMdbRMLOkITV54kax4WgadTdzUY
SBXNhOp_oqS0
```

Configure Spring Webflow encryption

CAS uses Spring Webflow to manage the authentication sequence, and this also needs to be encrypted. Add the following lines to `etc/cas/config/cas.properties`:

```
cas.webflow.crypto.signing.key:
cas.webflow.crypto.encryption.key:
```

Using the JSON Web Key Generator again (see above), generate an `HS256` key of size `512` and enter it for the value of the `cas.webflow.crypto.signing.key` property.

Unlike the ticket granting cookie encryption key above, the encryption key for Spring WebFlow is not a JSON Web Key. Rather, it's a randomly-generated string of 16 (by default) octets, Base64-encoded. An easy way to generate this key is to use `openssl`:

```
casdev-master# openssl rand -base64 16
Kmj1JJSPOTSiagI4gCxhUA==
casdev-master#
```

Enter the output from the `openssl` command for the value of the `cas.webflow.crypto.encryption.key` property. When finished, you should have something like this:

```
cas.webflow.crypto.signing.key:       hGapVlP6pCzIUo_CCboRszQpvWFPazm
yuWsBUOoWYqUQqMKw55al5c_EGH6VBtjpIVUqEAXcvLQjQ8HaVBEmDw
cas.webflow.crypto.encryption.key:    Kmj1JJSPOTSiagI4gCxhUA==
```

☑ **Tip:** The online JSON Web Key Generator is provided by the Mitre Corporation and the MIT Kerberos and Internet Trust Consortium, and is simply a web-based interface to the json-web-key-generator project, also provided by Mitre/MIT. The project can be cloned from GitHub and built locally if you don't trust the online generator, or you can download and use a pre-built copy from the CAS project by running the command

```
# curl -LO https://raw.githubusercontent.com/apereo/cas/master/etc/
jwk-gen.jar
```

Keys can then be generated using the command

```
# java -jar jwk-gen.jar -t oct -s [size]
```

References

- CAS 5: SSO Session Cookie
- CAS 5: Webflow Session

Configure logging settings

The Log4J configuration file included with the Maven WAR overlay template will attempt to write the CAS server log files (not the Tomcat log files) to the root of the CAS web application directory. However, since part of our Tomcat hardening procedure (page 39) includes removing write permission to this directory for the `tomcat` user, this will not work (and it's not a very good place for them anyway). So, just as we moved Tomcat's log files to `/var/log/tomcat`, we will move the CAS server's log files to `/var/log/cas`.

Edit the file `etc/cas/config/log4j2.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and find the line that defines the `cas.log.dir` property (around line 9) and change its value to `/var/log/cas`, like this:

```
<Property name="cas.log.dir" >/var/log/cas</Property>
```

Then create the `/var/log/cas` directory and set the ownership and permissions appropriately:

```
casdev-master# mkdir /var/log/cas
casdev-master# chown tomcat.tomcat /var/log/cas
casdev-master# chmod 750 /var/log/cas
```

Don't forget to run the three commands above on the individual CAS servers as well.

Adjust the log file rotation strategy (optional)

By default, the CAS log files will be rotated whenever their size reaches 10MB. On a busy server, this can result in numerous log files being created in a single day, making it more difficult to find particular events in the logs. To switch to a time-based rotation strategy in which the log files are rotated once a day, edit the `etc/cas/config/log4j2.xml` file again, and make the following changes:

1. In the `RollingFile` configuration for `cas.log` (around line 17), change the variable part of the `filePattern` attribute from `%d{yyyy-MM-dd-HH}-%i.log` to `%d{yyyy-MM-dd}.log` (remove the hour and sequence number from the pattern).
2. Remove (or comment out) the `OnStartupTriggeringPolicy` element

(around line 21).

3. Remove (or comment out) the `SizeBasedTriggeringPolicy` element (around line 22).
4. Add the attributes `interval="1" modulate="true"` to the `TimeBasedTriggeringPolicy` element (around line 23).

The end result should look like this:

```
<RollingFile name="file" fileName="${sys:cas.log.dir}/cas.log" appen
d="true"
             filePattern="${sys:cas.log.dir}/cas-%d{yyyy-MM-dd}.log">
    <PatternLayout pattern="%d %p [%c] - &lt;%m&gt;%n"/>
    <Policies>
        <TimeBasedTriggeringPolicy interval="1" modulate="true"/>
    </Policies>
</RollingFile>
```

Repeat the above changes for `cas_audit.log` (starting around line 26) and `perfStats.log` (starting around line 36).

❗ **Warning:** The configuration above assumes that there will be one, and only one, log file for each day. If a file with today's name already exists when Tomcat decides to rotate, the existing file will be ***overwritten***.

If you decide to keep the `OnStartupTriggeringPolicy` (which rotates the file whenever Tomcat starts) or the `SizeBasedTriggeringPolicy` (which rotates the file when it reaches a specified size (10MB by default)), or add some other policy, you should make sure the `filePattern` you use generates unique names if called more than once a day (e.g., by keeping the `%i` sequence number) or you will lose log data.

References

- CAS 5: Logging

Install and test the CAS application

To deploy the CAS application, we have to copy the application we just built with Maven into Tomcat's `webapps` directory and we have to copy the contents of the `etc/cas` directory to `/etc/cas`.

Create a distribution `tar` file

As explained in the section on hardening the Tomcat installation (page 39), web applications should be deployed as exploded directories rather than as WAR files, all files should be owned by user `root` and group `tomcat`, and file permissions should be set to owner read/write, group read only, and world none. To make it easier to accomplish all that, we will assemble everything into a single `tar` archive that can be copied to each CAS server and extracted. Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# tar czf /tmp/cassrv-files.tgz --owner=root --group=tom
cat --mode=g-w,o-rwx  etc/cas -C target cas --exclude cas/META-INF
```

to create the `tar` archive in `/tmp/cassrv-files.tgz`. The `--owner`, `--group`, and `--mode` options ensure that the files will have the correct owner, group, and permission settings when extracted. Since we will be running the above commands many times as we add more functionality to the server, it makes sense to put the above commands into a shell script (called, for example, `cassrv-tarball.sh`) like this:

```
#!/bin/sh

cd /opt/workspace/cas-overlay-template

tar czf /tmp/cassrv-files.tgz --owner=root --group=tomcat --mod
e=g-w,o-rwx \
    etc/cas -C target cas --exclude cas/META-INF

echo ""

ls -asl /tmp/cassrv-files.tgz
exit 0
```

Create an installation shell script

Because web application auto-deployment has been disabled as part of Tomcat server hardening, Tomcat has to be restarted when the application is updated. And to ensure that no out-of-date artifacts are left behind when installing a new version of the application, it's usually best to delete the old application directory rather than overwrite it. To make all this easier to do on multiple servers, all the commands can be collected into a shell script (called, for example, `/opt/scripts/cassrv-install.sh`) like this:

```
#!/bin/sh

echo "--- Installing on `hostname`"
umask 027

if [ -f /tmp/cassrv-files.tgz ]
then
    systemctl stop tomcat

    cd /
    rm -rf etc/cas/config
    tar xzf /tmp/cassrv-files.tgz etc/cas

    cd /opt/tomcat/latest/
    rm -rf webapps/cas work/Catalina/localhost/cas

    cd /opt/tomcat/latest/webapps
    tar xzf /tmp/cassrv-files.tgz cas

    systemctl start tomcat

    rm -f /tmp/cassrv-files.tgz /tmp/cassrv-install.sh
    echo "Installation complete."
else
    echo "Cannot find /tmp/cassrv-files.tgz; nothing installed."
    exit 1
fi

exit 0
```

This script will shut down Tomcat, delete the old contents of `/etc/cas` and extract a new set of files from the `tar` archive, delete the old copy of the application (and any associated runtime files) and extract a new copy from the `tar` archive, and then restart Tomcat.

Install and test on the master build server

Before distributing everything to the CAS servers, it should be tested on the master build server (***casdev-master***) to ensure that everything is working properly. To do this, run the installation script created above:

```
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the Tomcat log file (`/var/log/tomcat/catalina.yyyy-mm-dd.out`) for errors. All log messages in a successful start should be at log level `INFO`. If any messages are at log level `WARNING` or `SEVERE` (except for the "acceptable" warnings described in the Test the tomcat installation (page 59) section), then something is wrong and needs to be corrected.

There should be a line for the successful deployment of the `ROOT` web application, another for the successful deployment of the `CAS` web application, and finally a line for successful server startup:

```
DD-MMM-YYYY HH:MM:SS.sss INFO [localhost-startStop-1] org.apache.cata
lina.startup.HostConfig.deployDirectory Deployment of web applicatio
n directory [/var/lib/tomcat/ROOT] has finished in N ms
...
DD-MMM-YYYY HH:MM:SS.sss INFO [localhost-startStop-1] org.apache.cata
lina.startup.HostConfig.deployDirectory Deployment of web applicatio
n directory [/var/lib/tomcat/cas] has finished in N ms
...
DD-MMM-YYYY HH:MM:SS.sss INFO [main] org.apache.catalina.startup.Cata
lina.start Server startup in N ms
```

Then review the contents of the CAS log file (`/var/log/cas/cas.log`) for errors. For the most part everything should be at log level `INFO`, but there are a few `WARN` messages that will appear:

```
YYYY-MM-DD HH:MM:SS,sss WARN [org.apereo.cas.config.CasCoreTicketsCon
figuration] - <Runtime memory is used as the persistence storage for
retrieving and managing tickets. Tickets that are issued during runti
me will be LOST upon container restarts. This MAY impact SSO function
ality.>
YYYY-MM-DD HH:MM:SS,sss WARN [org.apereo.cas.config.support.authentic
ation.AcceptUsersAuthenticationEventExecutionPlanConfiguration] - <>
YYYY-MM-DD HH:MM:SS,sss WARN [org.apereo.cas.config.support.authentic
ation.AcceptUsersAuthenticationEventExecutionPlanConfiguration] - <


  ____    _____    ___    ____    _
 / ___|  |_   _|  / _ \  |  _ \  | |
 \___ \    | |   | | | | | |_) | | |
  ___) |   | |   | |_| | |  __/  |_|
 |____/    |_|    \___/  |_|     (_)


CAS is configured to accept a static list of credentials for authenti
cation. While this is generally useful for demo purposes, it is STRON
GLY recommended that you DISABLE this authentication method (by SETTI
NG 'cas.authn.accept.users' to a blank value) and switch to a mode th
at is more suitable for production.>
YYYY-MM-DD HH:MM:SS,sss WARN [org.apereo.cas.config.support.authentic
ation.AcceptUsersAuthenticationEventExecutionPlanConfiguration] - <>
YYYY-MM-DD HH:MM:SS,sss WARN [org.apereo.cas.config.CasCoreServicesCo
nfiguration] - <Runtime memory is used as the persistence storage fo
r retrieving and persisting service definitions. Changes that are mad
e to service definitions during runtime WILL be LOST upon container r
estarts.>
```

These are to be expected (and will be addressed in later steps of the deployment). If there are any other warnings or errors however, they should be corrected before proceeding.

Once everything has started, open up a web browser and enter the URL of the CAS application on the master build server:

```
https://casdev-master.newschool.edu:8443/cas/login
```

Expect the browser to complain about the TLS/SSL certificate because the host name of the server (***casdev-master.newschool.edu***) does not match the name in the certificate (***casdev.newschool.edu***). Click through the prompts to visit the site anyway, and you should be presented with a login page that looks something like this:

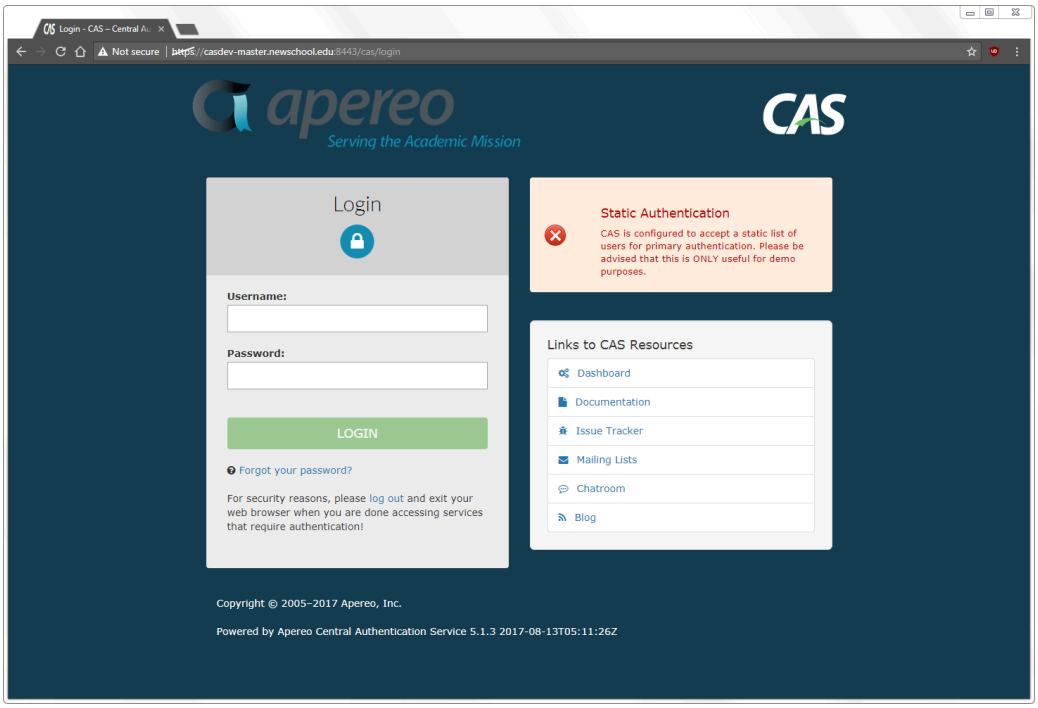

Figure 5. The basic CAS server login page

Since we have not configured the server with any authentication sources (yet), it comes with a set of built-in credentials for demonstration purposes. Log in using the username `casuser` and the password `Mellon` and you should then see a "successful login" page something like this:

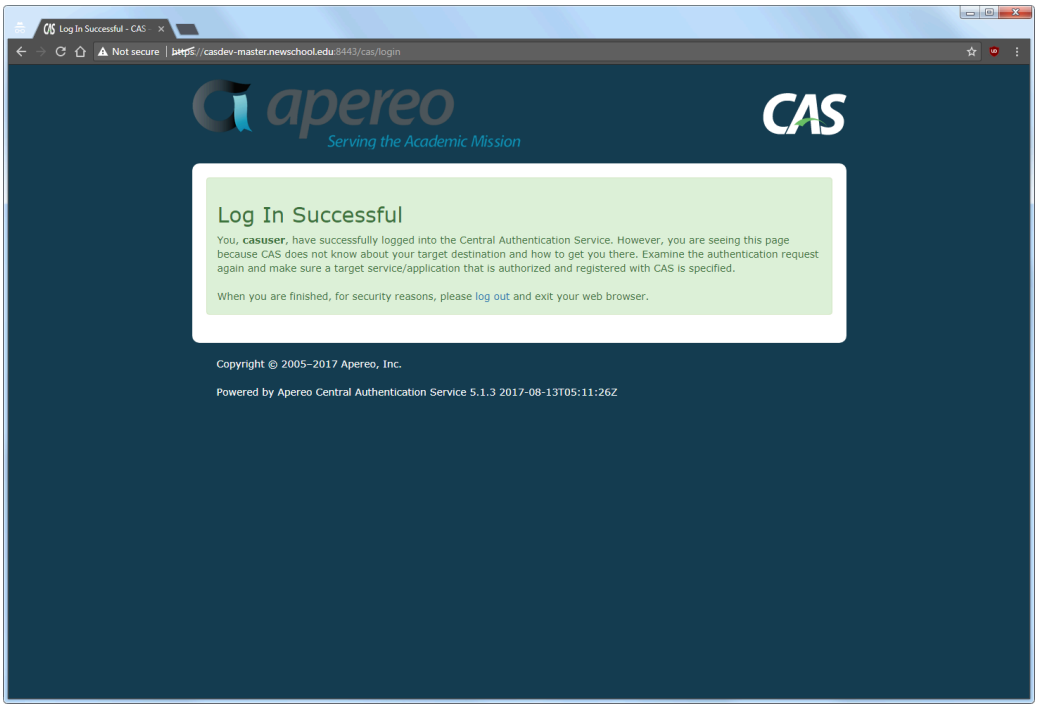

Figure 6. The CAS login success page

If this isn't what displays, check the various log files in `/var/log/tomcat` and `/var/log/cas` for errors.

Install and test on the CAS servers

Once CAS is running correctly on the master build server, it can be copied to the CAS servers using the `tar` archive and installation script created above. This can be done manually, or with a shell loop as shown below:

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Once all the servers have been updated, open up a web browser and enter the URL assigned to the load balancer's virtual interface:

```
https://casdev.newschool.edu/cas/login
```

Verify that the login page appears, and then enter the username and password (`casuser` / `Mellon`) and confirm that everything is working as it did on the master build server.

Define a CAS-specific service monitor on the load balancers

In Configure the load balancers (page 66), we defined a monitor for the server pool that connects to each server via HTTPS on port 8443 every 5 seconds and issues a `GET /` HTTP request. While this is sufficient to check whether or not the server itself is up and Tomcat is running, it's not sufficient to check that the CAS web

application is running. To do this, define a new monitor that issues a `GET /cas/login` request and checks for `Login - CAS` (part of the text on the login page) to be returned instead:

```
ltm monitor https /Common/casdev_https_8443_monitor {
    adaptive disabled
    cipherlist DEFAULT:+SHA:+3DES:+kEDH
    compatibility enabled
    defaults-from /Common/https
    description "Cas Dev Application HTTPS Monitor"
    destination *:8443
    interval 5
    is-dscp 0
    recv "Login - CAS"
    recv-disable none
    send "GET /cas/login\\r\\n"
    time-until-up 0
    timeout 16
}
```

And modify the pool definition to use that monitor instead:

```
ltm pool /Common/casdev_pool {
    description "CAS Development 8443 Pool"
    members {
        /Common/casdev-srv01:8443 {
            address 192.168.100.101
        }
        /Common/casdev-srv02:8443 {
            address 192.168.100.102
        }
        /Common/casdev-srv03:8443 {
            address 192.168.100.103
        }
    }
    monitor /Common/casdev_https_8443_monitor
}
```

Commit changes to Git

Before moving on to the next phase of configuration, commit the changes made so far to `log4j2.xml` and `cas.properties` to Git to make them easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git add etc/cas/config/log4j2.xml
casdev-master# git commit
```

on the master build server (***casdev-master***). The `git commit` command will bring up a text editor so you can describe the commit. Enter something like:

```
Basic server configuration:
 1. Set server host name and url information
 2. Configure TGC and webflow encryption
 3. Put log files into /var/log/cas
 4. Change log file rotation scheme
```

Then save and exit the editor, and Git will finish its work:

```
[newschool-casdev 63e0694] asic server configuration:  1. Set server
host name and url information  2. Configure TGC and webflow encryptio
n  3. Put log files into /var/log/cas  4. Change log file rotation sc
heme
 2 files changed, 42 insertions(+), 17 deletions(-)
 rewrite etc/cas/config/cas.properties (75%)
casdev-master#
```

Adding a service registry

Summary: A service registry must be added to the server so that client services can be declared and configured.

The CAS server includes a service management facility that allows CAS server administrators to declare and configure which services (CAS clients) may use the server, and how they may use it. The core component of the service management facility is the service registry that stores information about registered services including how the services must authenticate users, which users may access the service and under what conditions, data about authorized users the services may access, and so on.

The basic CAS server built in the previous section does not include a service registry (there is a line in `cas.properties` to enable a built-in registry, but it is commented out). Before we can build and use any test clients, it's necessary to add a service registry to the server. For our initial testing, we will add a simple registry that uses JSON files to describe services; we will replace this with a more robust registry when we configure the servers for high availability (page 321).

References

- CAS 5: Service Management

Add the feature and rebuild the server

Adding the JSON service registry feature requires adding a new dependency to the Maven project object model and rebuilding the server.

Add the JSON service registry to the project object model

To add JSON service registry support to the CAS server, edit the file `pom.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the `dependencies` section (around line 69), which should look something like this:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Insert a new dependency for the JSON service registry:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

Rebuild the server

Run Maven again to rebuild the server according to the new model:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output...check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 01:07 min
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 25M/70M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

References

- CAS 5: JSON Service Registry

Configure the service registry

Configuring the service registry requires defining the registry location in `cas.properties` and then creating service definition files for each service.

Define the service registry in `cas.properties`

Edit the file `etc/cas/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the commented-out definition of the service registry location (around line 7):

```
# cas.serviceRegistry.json.location: classpath:/services
```

Uncomment the line and change the property's value to `file:/etc/cas/services`:

```
cas.serviceRegistry.json.location:    file:/etc/cas/services
```

Create the service registry directory

Create the directory `etc/cas/services` in the `cas-overlay-template` directory on the master build server.

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# mkdir etc/cas/services
```

Create a service definition file

For simplicity (and to avoid worrying about the details of the service registry for the moment), create a "wildcard" service definition that will allow any HTTPS- or IMAPS-based service to make use of the CAS server. Create a file in the `etc/cas/services` directory on the master build server with the following contents:

```
{
  /*
   * Wildcard service definition that applies to any https or imaps u
rl.
   * Do not use this definition in a production environment.
   */
  "@class" :            "org.apereo.cas.services.RegexRegisteredServi
ce",
  "serviceId" :         "^(https|imaps)://.*",
  "name" :              "HTTPS and IMAPS wildcard",
  "id" :                1503925297,
  "evaluationOrder" :   99999
}
```

The CAS documentation recommends the following naming convention for JSON service definition files:

```
JSON filename = serviceName + "-" + serviceNumericId + ".json"
```

Therefore, the filename for the wildcard service definition above should be `HTTPSandIMAPSwildcard-1503925297.json`.

The CAS server uses Human JSON (Hjson), which relaxes JSON's strict syntax rules and also allows for the use of comments, to make it easier to write JSON service definitions by hand. (Later, we will build the management webapp (page 236) to maintain these files for us). The use of Hjson format for writing service definitions is optional; traditional JSON syntax is also supported.

The complete list of service definition properties is provided in the *Service Management* chapter of the CAS documentation, but the "interesting" fields in the definition above are:

`serviceId`	A regular expression describing the URL(s) where a service or services are located. Care should be taken to avoid patterns that match more than just the desired URL(s), as this can create security vulnerabilities.
`name`	A name for the service. Note that because the service definition filename is created based on this name (see above), the value of this field should never contain characters that are not allowed in filenames .

`id`	Unique numeric identifier for the service definition. An easy way to ensure that these identifiers are unique is to use the date and time the service definition was created. This can be represented as `YYYYMMDDhhmmss` or, for a more "anonymous" representation, as a timestamp (number of seconds since the epoch), which can be obtained with the command `date +%s` .
`evaluationOrder`	A value that determines the relative evaluation order of registered services (lower values come before higher values). This is especially important when more than one `serviceId` expression can match the same service; `evalutionOrder` deterines which expression is evaluated first.

References

- CAS 5: Service Management
- CAS 5: JSON Service Registry

Install and test the service registry

Before the service registry can be used, the rebuilt CAS application and the updated configuration files must be installed and tested.

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS application and configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Note: You may want to edit `cassrv-install.sh` and change the line that reads `rm -rf etc/cas/config` (around line 10) to read `rm -rf etc/cas/config etc/cas/services` instead, to ensure that repeated installations do not leave any old service definitions lying around.

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Commit changes to Git

Before moving on to building the CAS client, commit the changes made to `pom.xml` and `cas.properties`, as well as the new `etc/cas/services` directory, to Git to make them easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git add etc/cas/services
casdev-master# git add pom.xml
casdev-master# git commit -m "Added JSON service registry"
[newschool-casdev 5011d64] Added JSON service registry
 3 files changed, 17 insertions(+), 1 deletion(-)
 create mode 100644 etc/cas/services/wildcard.json
casdev-master#
```

on the master build server (***casdev-master***). The `git commit` command will not bring up a text editor as it did last time, since we provided the commit message on the command line.

Building the CAS client

Summary: To facilitate development and testing, a client application that interacts with the CAS server is needed.

Now that a basic CAS server is up and running, we can build a client application to talk to it.

Our CAS client will be an Apache HTTPD web server that offers both public content that anyone can access, and “secure” content that can only be accessed by authenticated and authorized users. The CAS server will be used to perform those authentication and authorization decisions.

Install the `mod_auth_cas` plugin

The `mod_auth_cas` plugin allows an Apache web server to interact with a CAS server via the CAS protocol. Red Hat does not offer this plugin for installation via `yum` however, so it must be downloaded and built from source code. We will build the plugin on the master build server (***casdev-master***) where the compilers and other development tools have been installed, and then copy it to the client server (***casdev-casapp***) for installation and use.

Install pre-requisites

The `mod_auth_cas` plugin build process depends on the presence of development libraries and header files from other packages. Run the commands

```
casdev-master# yum -y install httpd-devel
casdev-master# yum -y install openssl-devel
casdev-master# yum -y install libcurl-devel
```

to install them.

Clone the `mod_auth_cas` project

Use Git to clone the `mod_auth_cas` project from GitHub. Run the commands

```
casdev-master# cd /opt/workspace
casdev-master# git clone https://github.com/apereo/mod_auth_cas.git
Cloning into 'mod_auth_cas'...
remote: Counting objects: 1766, done.
remote: Total 1766 (delta 0), reused 0 (delta 0), pack-reused 1766
Receiving objects: 100% (1766/1766), 1.47 MiB | 0 bytes/s, done.
Resolving deltas: 100% (1060/1060), done.
casdev-master#
```

This will make a local copy of all files in the project and store them in a directory called `mod_auth_cas`. It will also record the information needed for Git to keep the local copy of the files synchronized with the copy stored on GitHub, so that corrections and updates made by the project team can be incorporated.

> ☑ **Tip:** As an alternative to using Git to clone a repository, GitHub allows the

files in a repository to be downloaded in a Zip archive. However, this method does not include the metadata that Git needs to keep the local copy in sync with the master repository.

Build the plugin

Run the commands

```
casdev-master# cd /opt/workspace/mod_auth_cas
casdev-master# autoreconf -ivf
(lots of output... check for errors)
casdev-master# ./configure
(lots of output... check for errors)
casdev-master# make
(lots of output... check for errors)
casdev-master#
```

to build the plugin.

Install the plugin on the client server

An Apache HTTPD plugin is really just a dynamic shared library that can be loaded at runtime. Run the commands

```
casdev-master# scp src/.libs/mod_auth_cas.so casdev-casapp:/etc/http
d/modules/mod_auth_cas.so
mod_auth_cas.so                               100%  241KB 240.7KB/
s   00:00
casdev-master# ssh casdev-casapp "chown root.root /etc/httpd/modules/
mod_auth_cas.so; chmod 755 /etc/httpd/modules/mod_auth_cas.so"
casdev-master#
```

to copy the `mod_auth_cas` module to the appropriate location on the server where Apache HTTP is installed (***casdev-casapp***).

References

- GitHub repo for `mod_auth_cas`

Configure HTTPD to use CAS

Now that the `mod_auth_cas` plugin has been built and installed, it can be configured, and some web content can be created to secure with it.

> ℹ **Note:** The steps in this section should be performed on the client server (***casdev-casapp***), not the master build server (***casdev-master***).

Configure `mod_auth_cas` settings

Create the file `/etc/httpd/conf.d/cas.conf` with the following contents to configure the `mod_auth_cas` module:

```
LoadModule auth_cas_module modules/mod_auth_cas.so

<Directory "/var/www/html/secured-by-cas">
    <IfModule mod_auth_cas.c>
        AuthType CAS
    </IfModule>

    Require valid-user
</Directory>

<IfModule mod_auth_cas.c>
    CASLoginUrl           https://casdev.newschool.edu/cas/login
    CASValidateUrl        https://casdev.newschool.edu/cas/serviceVal
idate
    CASCookiePath         /var/cache/httpd/mod_auth_cas/
    CASSSOEnabled         On
    CASDebug              Off
</IfModule>
```

If the CAS server is using a self-signed TLS/SSL certificate, the following line will also be needed:

```
    CASCertificatePath    /etc/pki/tls/certs/casdev.crt
```

and a copy of the public certificate should be installed in `/etc/pki/tls/certs/casdev.crt`.

Create the cookie cache directory

Run the commands

```
casdev-casapp# mkdir /var/cache/httpd/mod_auth_cas
casdev-casapp# chown apache.apache /var/cache/httpd/mod_auth_cas
casdev-casapp# chmod 700 /var/cache/httpd/mod_auth_cas
```

to create the directory specified in the `CASCookiePath` directive above.

Restart HTTPD

Run the command

```
casdev-casapp# systemctl restart httpd
```

to restart the HTTPD server with the new configuration. Check the log files in `/var/log/httpd` for errors.

Create example content

Edit the file `/var/www/html/index.php` and replace the call to `phpinfo()` with a link to another file, like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/
3.3.7/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <h1>Hello, World!</h1>
        <p><big>The quick brown fox jumped over the lazy dogs.</bi
g></p>
        <p><big>Click <a href="secured-by-cas/index.php">here</a> fo
r some secure content.</big></p>
    </div>
  </body>
</html>
```

Then create a directory, `/var/www/html/secured-by-cas`, and create the file `/var/www/html/secured-by-cas/index.php` with the following contents:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/
3.3.7/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <h1>Secured Content</h1>
      <p><big>This is some secure content. You should not be able to
see it until you have entered your username
        and password.</big></p>
      <h2>Attributes Returned by CAS</h2>
      <?php
        echo "<pre>";

        if (array_key_exists('REMOTE_USER', $_SERVER)) {
            echo "REMOTE_USER = " . $_SERVER['REMOTE_USER'] . "<br>";
        }

        $headers = getallheaders();
        foreach ($headers as $key => $value) {
            if (strpos($key, 'CAS_') === 0) {
                echo substr($key, 4) . " = " . $value . "<br>";
            }
        }

        echo "</pre>";
      ?>
    </div>
  </body>
</html>
```

The PHP code here will display environment variables and HTTP headers that are used by `mod_auth_cas` to pass attributes returned by the CAS server along to the web application.

Test the application

Now the use of CAS to protect the “secure” content created in the previous section can be tested by accessing the “public” part of the web site, and then clicking on the link to the “secure” section. At that point, the browser should be redirected to the CAS server, where a username and password can be entered. Provided that the username and password are correct, the secure content will be displayed.

Because both the load balancer and the CAS server use cookies, it’s usually best to perform testing with an “incognito” or “private browsing” instance of the web browser that deletes all cookies each time it is closed.

Shut down all but one of the pool servers

Operating CAS with a pool of servers instead of a single server requires special configuration. Because that configuration hasn’t been completed yet, testing must be performed against a single server. Therefore, the other servers in the pool should be shut down so that the load balancer will direct all traffic to that single server. Run the command

```
# systemctl stop tomcat
```

on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

Access the public site

Open up a web browser (in “incognito” or “private browsing” mode) and enter the URL of the CAS-enabled web site:

```
https://casdev-casapp.newschool.edu/
```

The contents of `/var/www/html/index.php` should be displayed, looking something like this:

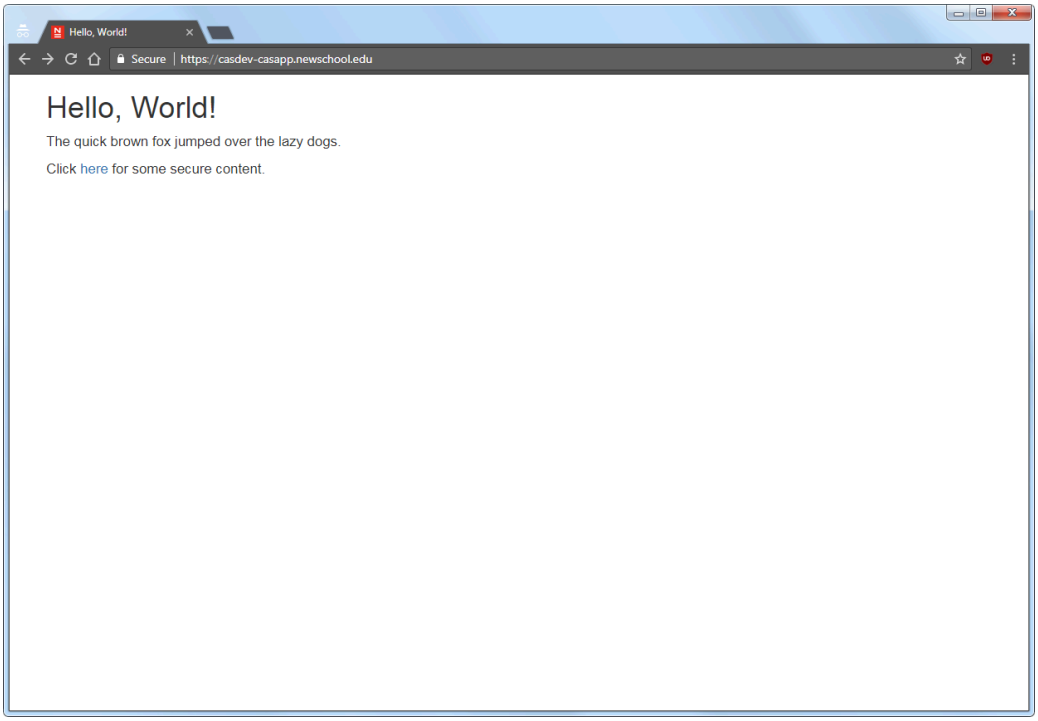

Figure 7. The "public" site

Access the secure area

Click on the “here” link to access the secure content, and you will be redirected to the CAS server login page, as shown below:

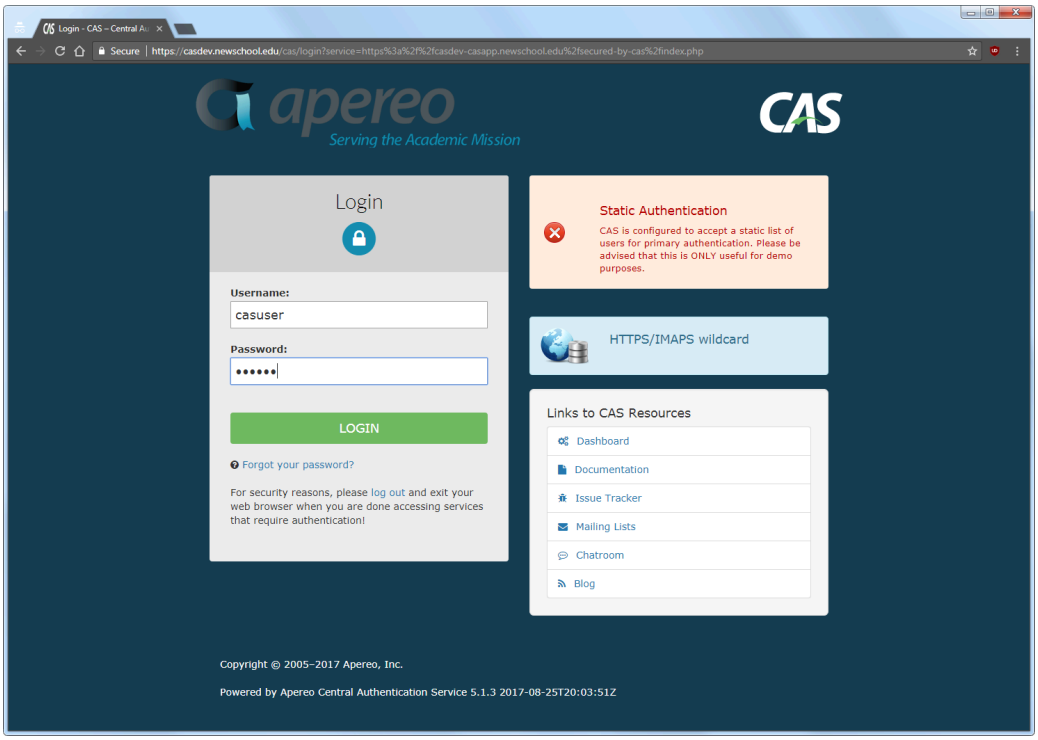

Figure 8. The CAS login page

Note that the contents of the `name` field from the service registry are displayed in the middle of the right-hand column. Enter a valid username and password (`casuser` / `Mellon`) and, upon successful authentication, the contents of `/var/www/html/secured-by-cas/index.php` will be displayed:

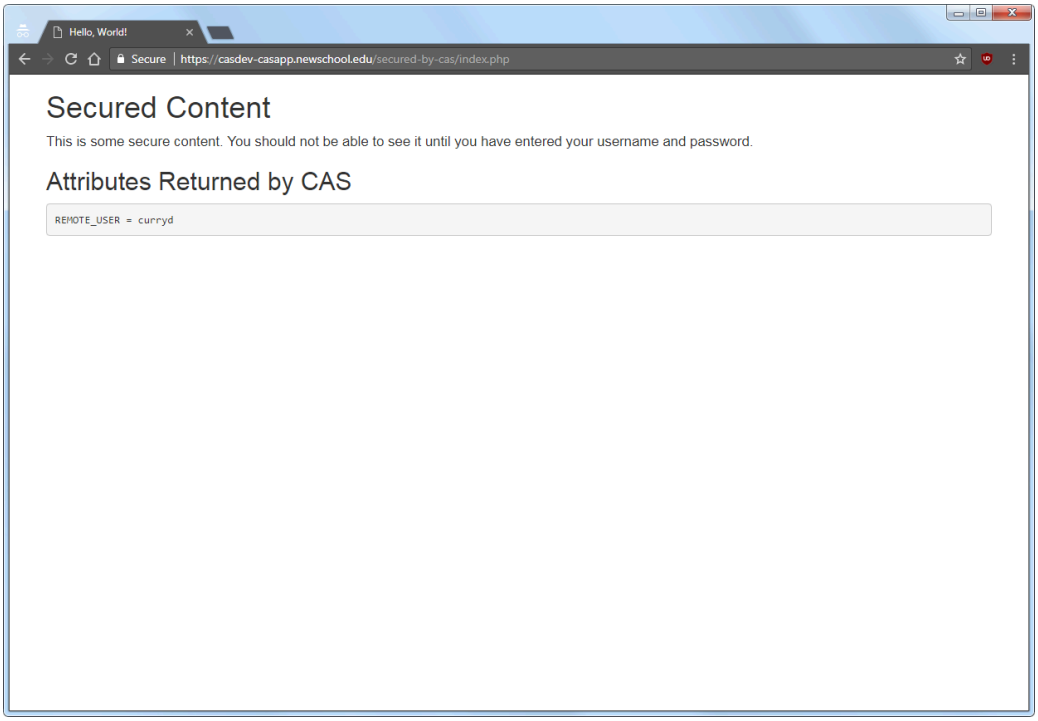

Figure 9. The "secure" content

Note that the value shown for the `REMOTE_USER` variable is the username that was entered on the CAS login page (`casuser`).

Restart the pool servers

One testing is complete, run the command

```
# systemctl start tomcat
```

on each of the pool servers shut down previously.

Adding LDAP support

Summary: Multiple LDAP directories will be used to authenticate users and to collect user attributes (ID numbers, names, group memberships, etc.) and make them available to client applications.

Although the default user credentials (`casuser` / `Mellon`) provided by the CAS server are useful for testing, we really want users to enter their own individual usernames and passwords. For the CAS server to support that in our environment, it has to be able to authenticate users against one or more LDAP directories. Many services that we use also require other information about users besides their username and password, such as their first and last name, student or employee ID numbers, group memberships, and so on. We store this information in LDAP as well, so the CAS server has to know how to retrieve it and send it to the client service.

In this section, we will add LDAP support to the CAS server to enable it to do three things:

1. **Authentication.** Prompt the user for his or her username and password, and validate that the provided password is indeed correct. At this stage, the user account is also checked to ensure that it is not suspended or disabled. At the conclusion of the authentication process, the CAS server will have identified a *security principal*. A CAS principal contains a unique identifier by which the authenticated user will be known to all requesting services. A principal also contains optional attributes that may be released to services to support authorization and personalization.
2. **Attribute resolution.** Specific attributes about the principal are collected from one or more sources and combined into a single set of attributes using any of several different combining strategies (merging, replacing, adding, etc.).
3. **Attribute release.** The process of defining how attributes are selected and provided to a given application in the final CAS response.

References

- CAS 5: Configuring Authentication Components
- CAS 5: Configuring Principal Resolution
- CAS 5: Attribute Resolution
- CAS 5: Attribute Release

Configuring LDAP authentication

Summary: The LDAP module will be added to the CAS server to enable it to authenticate users against LDAP directories

The CAS server's LDAP integration enables the server to authenticate users against LDAP directories such as Active Directory and the LDAP directory included with Ellucian's Luminis user portal.

Add the LDAP dependency to the project object model

To add LDAP support to the CAS server, edit the file `pom.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the dependencies section (around line 69), which should look something like this:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

Insert a new dependency for the LDAP module:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

This will instruct Maven to download the appropriate code modules and build them into the server.

Rebuild the server

Run Maven again to rebuild the server according to the new model:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 9.368 s
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 27M/78M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

Disable use of built-in credentials

Until they're disabled, CAS will use the built-in username and password (`casuser` / `Mellon`) regardless of what other authentication methods have been configured. To disable the built-in credentials, add the following line to `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***):

```
cas.authn.accept.users:
```

Commit changes to Git

Commit the changes made to `pom.xml` and `cas.properties` to Git to make them easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git add pom.xml
casdev-master# git commit -m "Added LDAP support"
[newschool-casdev 2dd8813] Added LDAP support
 2 files changed, 10 insertions(+)
casdev-master#
```

on the master build server (***casdev-master***).

References

- CAS 5: LDAP Authentication
- CAS 5: Configuration Properties: LDAP Authentication

Configure Active Directory authentication properties

Although CAS offers several dozen properties for controlling how LDAP authentication is performed, most of them come with reasonable defaults and do not have to be configured in normal circumstances. The complete list of properties can be found in the CAS documentation.

Add the following settings to `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) to authenticate against Active Directory:

```
cas.authn.ldap[0].order:                0
cas.authn.ldap[0].name:                 Active Directory
cas.authn.ldap[0].type:                 AD
cas.authn.ldap[0].ldapUrl:              ldaps://zuul.newschool.edu
cas.authn.ldap[0].validatePeriod:       270
cas.authn.ldap[0].poolPassivator:       NONE
cas.authn.ldap[0].userFilter:           sAMAccountName={user}
cas.authn.ldap[0].baseDn:               ou=TNSUsers,dc=tns,dc=newscho
ol,dc=edu
cas.authn.ldap[0].dnFormat:             cn=%s,ou=TNSUsers,dc=tns,dc=n
ewschool,dc=edu
```

The `[0]` in the property names indicates that this is the first LDAP source to be configured. Additional sources will be `[1]`, `[2]`, etc. (more on this in Configure Luminis LDAP authentication properties (page 136)).

The properties used above are:

`order`	When multiple authentication sources are configured, the CAS server looks for the user in one source after another until the user is found, and then the authentication is performed against that source (where it either succeeds or fails). This property influences the order in which the source is evaluated (if not specified, sources are evaluated in the order they are defined).
`name`	The name of the source. This is used when writing log file messages.
`type`	The type of authenticator to use. This should be `AD` for Active Directory.

`ldapUrl`	The URL of the Active Directory server. In our case, we use the URL of the virtual host on the F5 load balancer, which has multiple Active Directory servers behind it.
`validatePeriod`	The LDAP module periodically validates the connections in its connection pool. But the default setting for how often to do this (600 seconds) is longer than the idle timeout on the F5 load balancer that fronts the LDAP servers (300 seconds), which results in lots of warning messages being written to the CAS log file (one per connection every ten minutes). Reducing the validation period to something shorter than the load balancer idle timeout eliminates these messages.
`poolPassivator`	Passivators help manage LDAP connection pools. However, the default value for this property, `BIND`, does not work with the `AD` authenticator type, because there is no bind credential to use (the authenticator binds as the user being authenticated). Therefore, tihs setting is needed to disable the passivator.
`userFilter`	The LDAP filter to select the user from the directory. Active Directory typically searches on the `sAMAccountName` attribute. The `{user}` pattern will be replaced with the username string entered by the user.
`baseDn`	The base DN to search against when retrieving attributes. The "usual" value for this is more like `ou=Users,dc=example,dc=org`, but for historical reasons we keep our users in a different OU.
`dnFormat`	A format string to generate the user DN to be authenticated. In the string, `%s` will be replaced with the username entered on the login form. The "usual" value of this string is something more like `uid=%s,ou=Users,dc=example,dc=org`, but we do not use the `uid` attribute in our Active Directory schema, we use `cn` instead.

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS application and configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Once everything has started, open up a web browser and enter the URL of the CAS application on the master build server (`https://casdev-master.newschool.edu:8443/cas/login`), and try to log in using an Active Directory username and password. The “Log In Successful” page should appear. If it doesn’t, consult `/var/log/cas/cas.log` for errors.

It may help to enable debugging on the LDAP module by changing the `org.ldaptive` logging level to `debug` around line 95 in `/etc/cas/config/log4j2.xml` :

```
<AsyncLogger name="org.ldaptive" level="debug" />
```

and restarting Tomcat.

Install and test on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

and tested using the URL of the load balancer’s virtual interface (`https://casdev.newschool.edu/cas/login`).

Commit changes to Git

Before moving on to the next phase of configuration, commit the changes made to `pom.xml` and `cas.properties` to Git:

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git commit -m "Added Active Directory authentication"
[newschool-casdev 584aa7c] Added Active Directory authentication
 1 file changed, 16 insertions(+)
casdev-master#
```

References

- CAS 5: Configuration Properties: LDAP Authentication

Configure Luminis LDAP authentication properties

CAS allows multiple LDAP directories to be queried when authenticating users. Because our Active Directory does not include all our users, we also need to authenticate against the LDAP server included with Ellucian's Luminis user portal.

Add the following settings to `etc/cas/config/cas.properties`, below the Active Directory settings added in the previous section, to authenticate against Luminis LDAP:

```
cas.authn.ldap[1].order:                1
cas.authn.ldap[1].name:                 Luminis LDAP
cas.authn.ldap[1].type:                 AUTHENTICATED
cas.authn.ldap[1].ldapUrl:              ldaps://janus.newschool.edu
cas.authn.ldap[1].validatePeriod:       270
cas.authn.ldap[1].userFilter:           uid={user}
cas.authn.ldap[1].baseDn:               ou=People,o=cp
cas.authn.ldap[1].bindDn:               uid=ldap_ssotest,ou=Peopl
e,o=cp
cas.authn.ldap[1].bindCredential:       xxxxxxxxxxxx
```

The `[1]` in the property names indicates that this is the second LDAP source to be configured (Active Directory was `[0]`).

The properties used above are:

`order`	When multiple authentication sources are configured, the CAS server looks for the user in one source after another until the user is found, and then the authentication is performed against that source (where it either succeeds or fails). This property influences the order in which the source is evaluated (if not specified, sources are evaluated in the order they are defined).
`name`	The name of the source. This is used when writing log file messages.
`type`	The type of authenticator to use. This should be `AUTHENTICATED` to specify the traditional "bind account" method of authentication.

`ldapUrl`	The URL of the LDAP server. In our case, we use the URL of the virtual host on the F5 load balancer, which has multiple LDAP servers behind it.
`validatePeriod`	The LDAP module periodically validates the connections in its connection pool. But the default setting for how often to do this (600 seconds) is longer than the idle timeout on the F5 load balancer that fronts the LDAP servers (300 seconds), which results in lots of warning messages being written to the CAS log file (one per connection every ten minutes). Reducing the validation period to something shorter than the load balancer idle timeout eliminates these messages.
`userFilter`	The LDAP filter to select the user from the directory. Luminis LDAP searches on the `uid` attribute, which is actually the user's username. The `{user}` pattern will be replaced with the username string entered by the user.
`baseDn`	The base DN to search against when retrieving attributes.
`bindDN`	The DN of the account to bind to the directory with. This account must have search privileges on the directory.
`bindCredential`	The password to the bind account.

Install and test on the master build server

Adding the Luminis LDAP server only required changing `cas.properties`, so there is no need to rebuild or reinstall the server. Instead, just copy the new file into place on the master build server (***casdev-master***) and restart Tomcat by running the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# cp etc/cas/config/cas.properties /etc/cas/config/cas.p
roperties
casdev-master# systemctl restart tomcat
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Once everything has started, open up a web browser and enter the URL of the CAS application on the master build server (`https://casdev-master.newschool.edu:8443/cas/login`), and try to log in using a Luminis LDAP username and password (one that isn't also in Active Directory). The "Log In Successful" page should appear. If it doesn't, consult `/var/log/cas/cas.log` for errors. Then try logging in with an Active Directory username and password to confirm that the addition of LDAP didn't break anything.

It may help to enable debugging on the LDAP module by changing the `org.ldaptive` logging level to `debug` around line 95 in `/etc/cas/config/log4j2.xml` :

```
<AsyncLogger name="org.ldaptive" level="debug" />
```

and restarting Tomcat.

Install and test on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers:

```
casdev-master# for host in srv01 srv02 srv03
> do
> scp etc/cas/config/cas.properties casdev-${host}:/etc/cas/config/cas.properties
> ssh casdev-${host} systemctl restart tomcat
> done
casdev-master#
```

and tested using the URL of the load balancer's virtual interface (`https://casdev.newschool.edu/cas/login`).

Commit changes to Git

Before moving on to the next phase of configuration, commit the changes made to `pom.xml` and `cas.properties` to Git:

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git commit -m "Added Luminis LDAP authentication"
[newschool-casdev 912da34] Added Luminis LDAP authentication
 1 file changed, 15 insertions(+)
casdev-master#
```

References

- CAS 5: Configuration Properties: LDAP Authentication

Configuring LDAP attribute resolution and release

Summary: To enable client applications to obtain information about authenticated users, the CAS server must be configured to resolve attributes and release them to the clients.

Version 3 of the CAS protocol, which was first supported by CAS 4.0, contains native support for returning authentication/user attributes to clients. Version 2 of the CAS protocol, the version implemented by CAS 3.*x*, did not support attribute release; the SAML 1.1 protocol was used for that purpose. Most CAS clients have not yet been updated to support Version 3 of the protocol, so it's still necessary to configure SAML 1.1-based attribute release.

Add the SAML 1.1 dependency to the project object model

To add SAML 1.1 support to the CAS server, edit the file `pom.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the dependencies section (around line 69), which should look something like this:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

Insert a new dependency for the SAML 1.1 module:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

This will instruct Maven to download the appropriate code modules and build them into the server.

Rebuild the server

Run Maven again to rebuild the server according to the new model:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 43.966 s
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 30M/76M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

References

- CAS 5: SAML Protocol

Configure attribute resolution

Now that the server has been configured to support attribute release, it must be configured to resolve (retrieve) the attributes to be released. Since the LDAP module has already been added to the server, all that is necessary to enable this is the definition of some additional properties.

Configure Active Directory attribute resolution

Add the following lines to `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) to enable CAS to resolve attributes from Active Directory:

```
cas.authn.attributeRepository.ldap[0].order:            0
cas.authn.attributeRepository.ldap[0].ldapUrl:          ldaps://zuu
l.newschool.edu
cas.authn.attributeRepository.ldap[0].validatePeriod:   270
cas.authn.attributeRepository.ldap[0].userFilter:       sAMAccountNam
e={user}
cas.authn.attributeRepository.ldap[0].baseDn:           ou=TNSUsers,d
c=tns,dc=newschool,dc=edu
cas.authn.attributeRepository.ldap[0].bindDn:           cn=ldap_ssote
st,ou=Service,ou=Users,ou=Enterprise Support,dc=tns,dc=newschool,dc=e
du
cas.authn.attributeRepository.ldap[0].bindCredential:   xxxxxxxxxxx
cas.authn.attributeRepository.ldap[0].attributes.cn:    uid
cas.authn.attributeRepository.ldap[0].attributes.displayName:   displ
ayName
cas.authn.attributeRepository.ldap[0].attributes.givenName:     given
Name
cas.authn.attributeRepository.ldap[0].attributes.mail:  mail
cas.authn.attributeRepository.ldap[0].attributes.sn:    sn
cas.authn.attributeRepository.ldap[0].attributes.tnsGoogleAppsRol
e:     role
cas.authn.attributeRepository.ldap[0].attributes.tnsIDNumber:   cn
```

The first seven properties should be self-explanatory (or see the descriptions in the previous sections). Note that while we did not need to use a bind account to authenticate users against Active Directory, we do need to use one to resolve attributes.

The `.attributes.` properties specify, for each attribute, its name in the directory, and the name it should be given when sending it to the client application (the *mapped* name). For example, in the set of attributes above, the Active Directory attributes `cn`, `displayName`, `givenName`, `mail`, `sn`, `tnsGoogleAppsRole`, and `tnsIDNumber` will be retrieved and may be sent to client applications. The attributes named `cn`, `tnsGoogleAppsRole`, and `tnsIDNumber` will be released with the mapped names `uid`, `role`, and `cn` respectively, while the other attributes' names will not be changed.

Configure Luminis LDAP attribute resolution

Add the following lines to `etc/cas/config/cas.properties` to enable CAS to resolve attributes from Luminis LDAP:

```
cas.authn.attributeRepository.ldap[1].order=            1
cas.authn.attributeRepository.ldap[1].ldapUrl=          ldaps://janu
s.newschool.edu
cas.authn.attributeRepository.ldap[1].validatePeriod=   270
cas.authn.attributeRepository.ldap[1].userFilter=       uid={user}
cas.authn.attributeRepository.ldap[1].baseDn=           ou=People,o=c
p
cas.authn.attributeRepository.ldap[1].bindDn=           uid=ldap_ssot
est,ou=People,o=cp
cas.authn.attributeRepository.ldap[1].bindCredential=   xxxxxxxxxxxx
cas.authn.attributeRepository.ldap[1].attributes.cn=    cn
cas.authn.attributeRepository.ldap[1].attributes.displayName=   displ
ayName
cas.authn.attributeRepository.ldap[1].attributes.givenName=     given
Name
cas.authn.attributeRepository.ldap[1].attributes.mail=  mail
cas.authn.attributeRepository.ldap[1].attributes.sn=    sn
cas.authn.attributeRepository.ldap[1].attributes.udcid= UDC_IDENTIFIE
R
cas.authn.attributeRepository.ldap[1].attributes.uid=   uid
```

As above, the first seven properties should be self-explanatory. The list of attributes to be released is similar to, but not the same as, the list for Active Directory, above.

One difference is that the two directories use different attributes for the same information. Luminis LDAP stores the username in the `uid` attribute and the student/employee ID number in the `cn` attribute. Active Directory on the other hand, stores the username in the `cn` attribute, and stores the student/employee ID number in a custom attribute called `tnsIDNumber`. To make things match up so the

same data is in the same attribute from both directories (the reason for this will become apparent below), the Active Directory configuration above switches things around to match Luminis LDAP by mapping `cn` to `uid` and `tnsIDNumber` as `cn`.

Another difference is that Active Directory has an attribute called `tnsGoogleAppsRole` (released as `role`) that Luminis LDAP doesn't have, and Luminis LDAP has an attribute called `udcid` (released as `UDC_IDENTIFIER`) that Active Directory doesn't have.

Configure an attribute merging strategy

Although CAS will only authenticate a user against the first directory (according to the evaluation order) in which the user is found, it will attempt to retrieve attributes from all configured repositories and then merge them together. The *merging strategy* determines what happens when CAS discovers the same attribute (based on the mapped name) in multiple repositories. The options are:

`REPLACE`	Overwrites the existing value (if any) with the new value. The attribute will contain the last value discovered.
`ADD`	Retain the existing value (if any), and ignore any subsequent values discovered for the same attribute. The attribute will contain the first value discovered.
`MERGE`	Combine all values into a single attribute, resulting in a comma-separated list of values.

In our case, we have a mix of users who are only in Active Directory, users who are only in Luminis LDAP, and users who are in both directories. Most of the time the duplicated attributes have the same value in both directories, but there are just enough exceptions to make `MERGE` a bad idea (applications that don't expect to receive multi-valued attributes don't handle them well). We have therefore (somewhat arbitrarily) decided that for users in both directories, the values of their attributes in Active Directory should "win," and since Active Directory is the first repository, we want to use the `ADD` strategy. So, add the following line to `etc/cas/config/cas.properties`:

```
cas.authn.attributeRepository.merger:                    ADD
```

Had we instead decided that Luminis LDAP should "win," `REPLACE` would be the correct strategy. Or, we could stick with the `ADD` strategy and change the evaluation order of the repositories.

References

- CAS 5: Configuration Properties: Authentication Attributes
- CAS 5: Configuration Properties: Merging Strategies

Update the service registry

Attribute release policies are defined on a per-service basis in the service registry. There are four basic attribute release policies:

Return All	Return all resolved attributes to the service.
Deny All	Do not return any attributes to the service. This will also prevent the release of the default attribute pool (see the note below).
Return Allowed	Only return the attributes specifically allowed by the policy. This policy includes a list of the attributes to release.
Return Mapped	Only return the attributes specifically allowed by the policy, but also allow them to be renamed at the individual service level. Useful when a particular service insists on having specific attribute names not used by other services.

The syntax for defining the above policies is defined in the CAS 5 *Attribute Release Policies* documentation. That document also describes a number of script-based policies that will call a Groovy, JavaScript, or Python script to decide how to release attributes (these policies are beyond the scope of this document).

> ℹ **Note:** The `cas.authn.attributeRepository.defaultAttributesToRelease` property can be set in `cas.properties` to a comma-separated list of attributes that should be released to all services, without having to list them in every service definition. We are not using this feature in our installation, because it makes it harder to determine which attributes are released to a particular service (by requiring the administrator to look in more than one location).

Create a “return all attributes” service definition for the CAS client

When we initially created the service registry (page 110), we created a wildcard service definition that would match any service. Now however, it makes sense to create a specific definition for our CAS client, and use that definition to release attributes to the client. Create a file in the `etc/cas/services` directory on the master build server (***casdev-master***) with the following contents:

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^https://casdev-casapp.newschool.edu/secured-by-ca
s(\\z|/.*)",
  "name" : "Apache Secured By CAS",
  "id" : 1504122840,
  "description" : "CAS development Apache mod_auth_cas server with us
ername/password protection",
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePoli
cy"
  },
  "evaluationOrder" : 1100
}
```

Following the naming convention introduced earlier (page 110), the name of this file should be `ApacheSecuredByCAS-1504122840.json`. The `id` value, and therefore that part of the filename, should use the current date and time represented as a timestamp (as output by `date +%s`) or as `YYYYMMDDhhmmss`.

This service definition uses a `serviceId` regular expression that matches only the URL for the `secured-by-cas` directory on the ***casdev-casapp*** server. The `(\\z|/.*)` syntax at the end matches either the empty string (`\\z`) or a slash ('/') followed by anything (`/.*`), meaning that the following will match:

```
https://casdev-casapp.newschool.edu/secured-by-cas
https://casdev-casapp.newschool.edu/secured-by-cas/
https://casdev-casapp.newschool.edu/secured-by-cas/index.php
https://casdev-casapp.newschool.edu/secured-by-cas/subdir/file.html
```

but the following will not:

```
https://casdev-casapp.newschool.edu/secured-by-cas-and-something-else
https://casdev-casapp.newschool.edu/some/other/path
https://casdev-master.newschool.edu/secured-by-cas
```

This service definition uses a `description` property instead of a comment to describe the service; this way the definition will appear in the management webapp (page 236).

The `evaluationOrder` has been given a value lower than that of the wildcard definition, so this definition will be matched first.

And finally, this definition includes the “Release All” `attributeReleasePolicy` property, which means that the CAS client will receive all attributes that could be resolved for the authenticating user.

Create a “return mapped attributes” service definition for the CAS client

One of the applications that we use, Ellucian’s Luminis portal, expects to receive a couple of attributes with names other than the ones commonly used: instead of a `mail` attribute, it expects to receive an `EmailAddress` attribute, and instead of a `givenName` attribute, it expects to receive a `Formatted Name` attribute (despite the fact that attribute names are not supposed to contain spaces). We could have made these mappings in the `cas.properties` file, but that would then require all applications to support these unusual attribute names. So instead, we will use the “Return Mapped” attribute release policy to perform the mapping only for this application.

To test this idea with our CAS client, create a file in the `etc/cas/services` directory on the master build server (***casdev-master***) with the following contents:

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^https://casdev-casapp.newschool.edu/return-mappe
d(\\z|/.*)",
  "name" : "Return Mapped Test",
  "id" : 1506518400,
  "description" : "Display results of a Return Mapped attribute relea
se policy",
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnMappedAttributeReleaseP
olicy",
    "allowedAttributes" : {
      "@class" : "java.util.TreeMap",
      "cn" : "cn",
      "displayName" : "displayName",
      "givenName" : "Formatted Name",
      "mail" : "EmailAddress",
      "memberOf" : "memberOf",
      "role" : "role",
      "sn" : "sn",
      "uid" : "uid",
      "UDC_IDENTIFIER": "UDC_IDENTIFIER"
    }
  },
  "evaluationOrder" : 1150
}
```

The name of this file should be `ReturnMappedTest-1506518400.json`. The `id` value, and therefore that part of the filename, should use the current date and time represented as a timestamp (as output by `date +%s`) or as `YYYYMMDDhhmmss`.

In this definition, the `attributeReleasePolicy` property uses the `ReturnMappedAttributeReleasePolicy` instead of the `ReturnAllAttributeReleasePolicy`; this requires us to provide a new sub-property called `allowedAttributes` that contains the list of attributes to be released. For each attribute, the attribute's name (as set in `cas.properties`) appears on the left, and the name it should be released with (the *mapped* name) *for this application only* appears on the right.

References

- CAS 5: Attribute Release Policies
- CAS 5: JSON Service Registry

Update the CAS client configuration

Now that the CAS server has been configured to resolve attributes and release them to the CAS client, the CAS client has to be configured to ask for them.

Update `mod_auth_cas` settings

Edit the file `/etc/httpd/conf.d/cas.conf` on the client server (***casdev-casapp***) and make the following changes:

1. In the `<Directory>` directive, add a line to set `CASAuthNHeader` to `On`. This tells `mod_auth_cas` to add an HTTP header containing the user returned by CAS.
2. Add a second `<Directory>` directive, just like the first, except using the path `/var/www/html/return-mapped`
3. At the bottom of the file, change the value of the `CASValidateURL` setting from `.../serviceValidate` to `.../samlValidate`. This is the endpoint provided by the server for authentcating users and returning attribues via SAML 1.1.
4. At the bottom of the file, add a line to set `CASValidateSAML` to `On`. This tells `mod_auth_cas` to use SAML 1.1 to retrieve user attributes and store them as HTTP headers.

The result should look like this:

```
LoadModule auth_cas_module modules/mod_auth_cas.so

<Directory "/var/www/html/secured-by-cas">
    <IfModule mod_auth_cas.c>
        AuthType        CAS
        CASAuthNHeader  On
    </IfModule>

    Require valid-user
</Directory>

<Directory "/var/www/html/return-mapped">
    <IfModule mod_auth_cas.c>
        AuthType        CAS
        CASAuthNHeader  On
    </IfModule>

    Require valid-user
</Directory>

<IfModule mod_auth_cas.c>
    CASLoginUrl           https://casdev.newschool.edu/cas/login
    CASValidateUrl        https://casdev.newschool.edu/cas/samlValida
te
    CASCookiePath         /var/cache/httpd/mod_auth_cas/
    CASValidateSAML       On
    CASSSOEnabled         On
    CASDebug              Off
</IfModule>
```

Create a new secure content area

Make a copy of the existing secure content area on the client server (***casdev-casapp***):

```
casdev-casapp# cd /var/www/html
casdev-casapp# cp -rp secured-by-cas return-mapped
```

Then edit the file `return-mapped/index.html` and update the heading and paragraph of text to reflect the requirements to view it:

```
<h1>Return Mapped Attributes</h1>
<p><big>This is some secure content. You should not be able to see it
 until you have entered your username and password. The attributes in
 the list below should have their "new" names as a result of using a
 "Return Mapped" attribute release policy.</big></p>
```

Leave the rest of the file unchanged.

Update the public content page

Update `/var/www/html/index.php` to include a link to the new secure area:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1">
    <link rel="stylesheet"
      href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.m
in.css">
  </head>
  <body>
    <div class="container">
      <h1>Hello, World!</h1>
        <p><big>The quick brown fox jumped over the lazy dogs.</bi
g></p>
        <p><big>Click <a href="secured-by-cas/index.php">here</a> fo
r some
          content secured by username and password.</big></p>
        <p><big>Click <a href="return-mapped/index.php">here</a> to s
ee the
          results of the "Return Mapped" attribute release policy.</b
ig></p>
    </div>
  </body>
</html>
```

Restart HTTPD

Run the command

```
casdev-casapp# systemctl restart httpd
```

to restart the HTTPD server with the new configuration. Check the log files in `/var/log/httpd` for errors.

References

- GitHub repo for `mod_auth_cas`

Install and test the application

Before attribute resolution and release can be used, the rebuilt CAS application and the updated configuration files must be installed. Then everything can be tested by accessing the “public” part of the client application web site and then clicking on the link to the “secure” section. At that point, the browser should be redirected to the CAS server, where upon successful authentication the secure content, including the values of the attributes, will be displayed.

Because both the load balancer and the CAS server use cookies, it’s usually best to perform testing with an “incognito” or “private browsing” instance of the web browser that deletes all cookies each time it is closed.

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS application and configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Shut down all but one of the pool servers

Operating CAS with a pool of servers instead of a single server requires special configuration. Because that configuration hasn't been completed yet, testing must be performed against a single server. Therefore, the other servers in the pool should be shut down so that the load balancer will direct all traffic to that single server. Run the command

```
# systemctl stop tomcat
```

on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

Access the public site

Open up a web browser (in "incognito" or "private browsing" mode) and enter the URL of the CAS-enabled web site:

```
https://casdev-casapp.newschool.edu/
```

The contents of `/var/www/html/index.php` should be displayed, looking something like this:

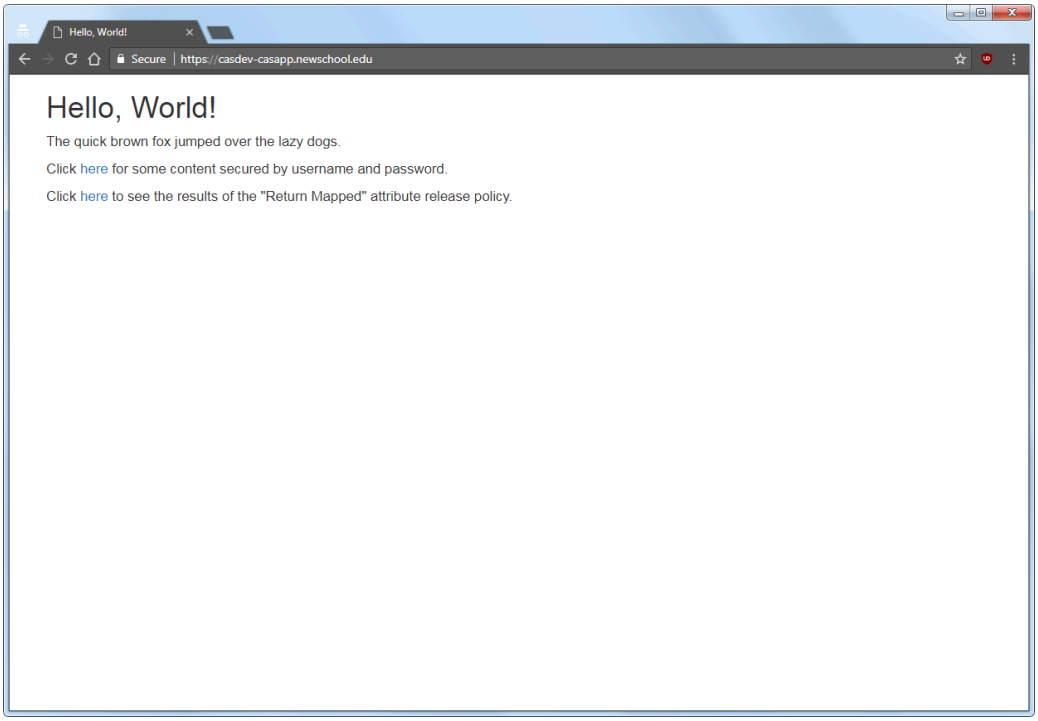

Figure 10. The "public" site

Access the “return all attributes” secure area

Click on the first “here” link to access the “release all attributes” secure content, and you will be redirected to the CAS server login page, as shown below:

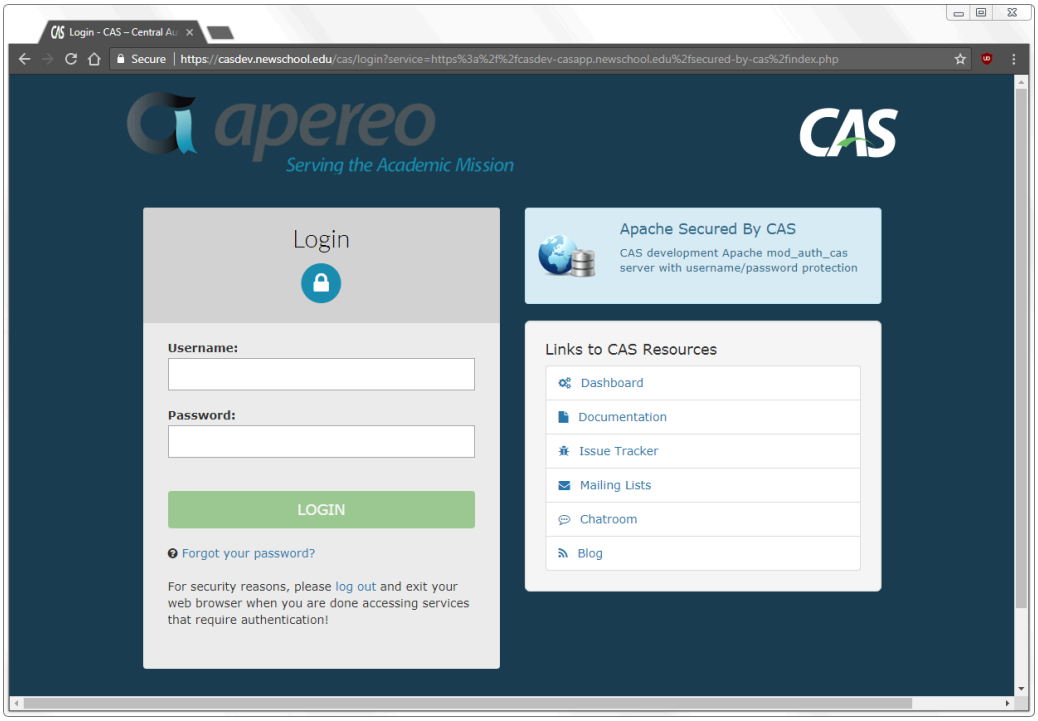

Figure 11. The CAS login page

Note that the contents of the `name` field from the service registry are displayed at the top of the right-hand column; make sure that `Apache Secured by CAS` is displayed here and not `HTTPS and IMAPS wildcard`. Enter a valid username and password (Active Directory or LDAP) and, upon successful authentication, the contents of `/var/www/html/secured-by-cas/index.php` will be displayed:

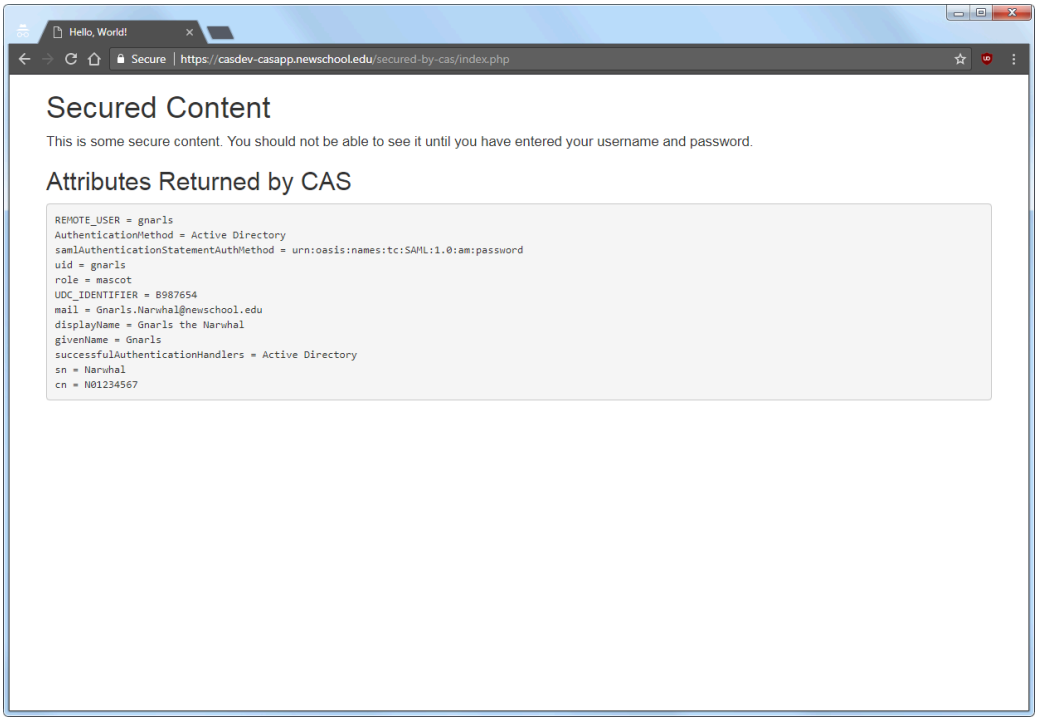

Figure 12. The "secure" content

Note that the value shown for the `REMOTE_USER` variable is the username that was entered on the CAS login page (`gnarls`), and that the attributes configured for release (page 144) are all shown, along with some additional values provided as part of the SAML 1.1 response from the CAS server.

Access the “return mapped attributes” secure area

Click the “back” button in the browser a couple of times (or start up a new one in “incognito” or “private browsing” mode) to get back to the “public” page shown in Figure 10, and then click on the second “here” link to access the “return mapped attributes” secure area, which will display the contents of `/var/www/html/return-mapped/index.php` :

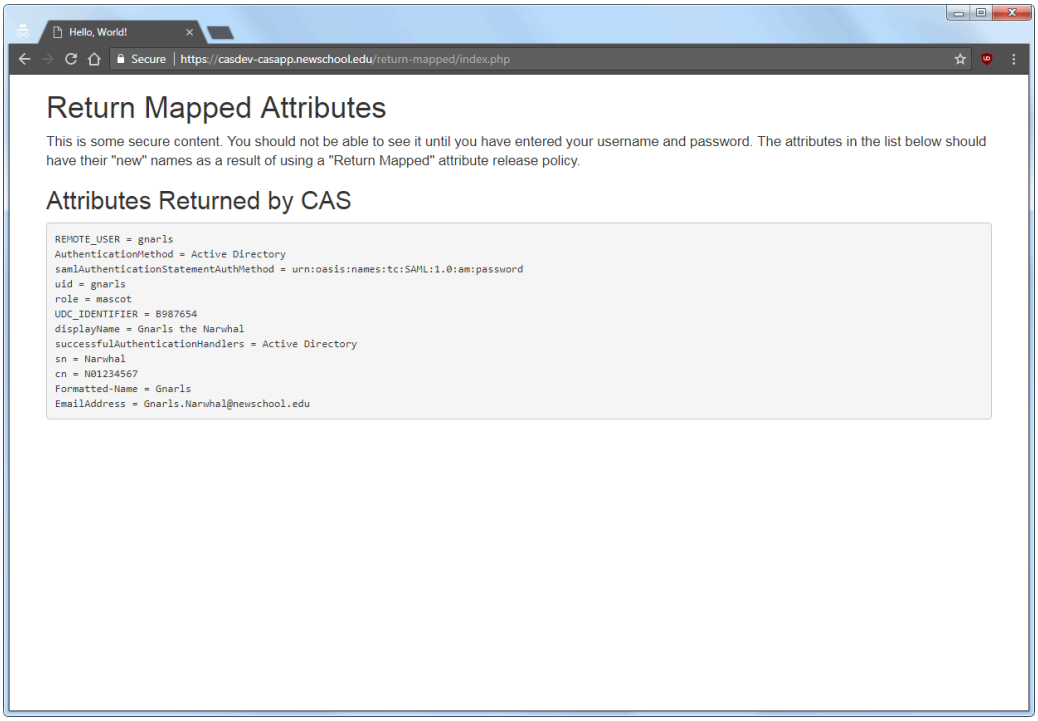

Figure 13. The "mapped" content

Note that the `mail` and `givenName` attributes have been replaced with `EmailAddress` and `Formatted-Name` (spaces in attributed names are not allowed, so a '-' is inserted), but the values of the attributes are the same.

Restart the pool servers

One testing is complete, run the command

```
# systemctl start tomcat
```

on each of the pool servers shut down previously.

Commit changes to Git

Before moving on to to the next phase of configuration, commit the changes made to `pom.xml` and `cas.properties`, as well as the new `etc/cas/services/ApacheSecuredByCAS-1504122840.json` and `etc/cas/services/ReturnMappedTest-1506518400.json` files, to Git to make them easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git add etc/cas/services/ApacheSecuredByCAS-150412284
0.json
casdev-master# git add etc/cas/services/ReturnMappedTest-1506518400.j
son
casdev-master# git add pom.xml
casdev-master# git commit
```

on the master build server (***casdev-master***). The `git commit` command will will bring up a text editor so you can describe the commit. Enter something like:

```
Add LDAP support:
 1. Add LDAP and SAML 1.1 modules to server
 2. Configure Active Directory authentication
 3. Configure Luminis LDAP authentication
 4. Configure AD/LDAP attribute resolution
 5. Create CasApp service definition
```

Then save and exit the editor, and Git will finish its work:

```
[newschool-casdev 0cb7e85] Add LDAP support:  1. Add LDAP and SAML
1.1 modules to server  2. Configure Active Directory authentication
3. Configure Luminis LDAP authentication  4. Configure AD/LDAP attrib
ute resolution  5. Create CasApp service definition
 3 files changed, 64 insertions(+)
 create mode 100644 etc/cas/services/ApacheSecuredByCAS-1504122840.js
on
 create mode 100644 etc/cas/services/ReturnMappedTest-1506518400.json
casdev-master#
```

Adding MFA support

Summary: Multi-factor authentication from Duo Security will be used to secure access to applications containing sensitive information or providing sensitive functionality.

CAS 5 provides a flexible framework for multi-factor authentication (MFA) that supports multiple multi-factor providers. MFA can be required on a per-service basis or across the board for all services. It can be required for individual named users, groups of users, or all users. Multiple MFA products/solutions can be supported in the same CAS server instance (and indeed, if desired, multiple MFA products/solutions can be required to access a single service).

The New School is currently in the early stages of rolling out Duo Security to all faculty and staff to access certain select applications. Duo offers several options for authenticating users:

- a mobile push notification and one-button verification of identity to a smartphone (requires the free Duo Mobile app)
- a one-time code generated on a smartphone
- a one-time code generated by Duo and sent to a handset via SMS text messaging
- a telephone call from that will prompt you to validate the login request

Add the Duo dependency to the project object model

To add Duo support to the CAS server, edit the file `pom.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the dependencies section (around line 69), which should look something like this:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

Insert a new dependency for the Duo module:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
         <groupId>org.apereo.cas</groupId>
         <artifactId>cas-server-support-duo</artifactId>
         <version>${cas.version}</version>
    </dependency>
</dependencies>
```

This will instruct Maven to download the appropriate code modules and build them into the server.

Rebuild the server

Run Maven again to rebuild the server according to the new model:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 01:02 min
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 30M/79M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

References

- CAS 5: Multi-factor Authentication
- CAS 5: Duo Security Authentication

Configure Duo authentication

Configuring Duo authentication requires setting up a new application to be protected via the Duo administrator console, and then copying some of the information from that configuration to the `cas.properties` file.

Create a new Duo protected application

To create a new Duo protected application:

1. Log into the Duo administrator console.
2. Select “Applications > Protect an Application” from the links on the left side of the window.
3. Find the entry for **CAS (Central Authentication Service)** in the list of applications and click on “Protect this Application”.
4. Scroll down to the **Settings** section and set the application name to something meaningful, for example, `CASDev CAS Server`.
5. Make any other settings changes as appropriate.
6. Click “Save Changes”.

Don’t log out of the administrator console yet; some of the information there must be copied to `cas.properties`.

Configure Duo authentication properties

Add the following settings to `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) to enable Duo authentication:

```
cas.authn.mfa.duo[0].duoApiHost:        api-a1b2c3d4.duosecurity.com
cas.authn.mfa.duo[0].duoIntegrationKey: DIYQCAFU5Q5UCD24J00R
cas.authn.mfa.duo[0].duoSecretKey:      FeTtpcOFyDxvtrtOXqma74DztXf7I
NRDKENMhAOF
cas.authn.mfa.duo[0].duoApplicationKey: 3d787231f9b9e128a9b94647b6e96
8f1fe0deddc
```

The `[0]` in the property names indicates that this is the first Duo provider to be configured. Additional providers (for example, if there are different providers for different locations or different groups of users) will be `[1]`, `[2]`, etc.

The `duoApiHost`, `duoIntegrationKey`, and `duoSecretKey` values should be copied from the **Details** section of the protected application page in the Duo administrator console (see above).

The `duoApplicationKey` value is a string, at least 40 characters long, that is generated locally and is *not* shared with Duo. The CAS documentation offers the procedure below for generating this string:

```
casdev-master# python
Python 2.7.5 (default, Aug  2 2016, 04:20:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more informatio
n.
>>> import os, hashlib
>>> print hashlib.sha1(os.urandom(32)).hexdigest()
3d787231f9b9e128a9b94647b6e968f1fe0deddc
>>> exit()
casdev-master#
```

References

- CAS 5: Configuration Properties: Duo Security

Update the CAS client configuration

To allow the CAS client to test/demonstrate both content secured only by CAS and content secured by both CAS and Duo at the same time, create a new secure content area on the CAS client server and configure Apache HTTPD to protect it.

Create a new secure content area

Make a copy of the existing secure content area on the client server (***casdev-casapp***):

```
casdev-casapp# cd /var/www/html
casdev-casapp# cp -rp secured-by-cas secured-by-cas-duo
```

Then edit the file `secured-by-cas-duo/index.php` and update the paragraph of text to reflect the requirements to view it:

```
<p><big>This is some secure content. You should not be able to see it
 until you have entered your username and password and authenticated
 with Duo.</big></p>
```

Leave the rest of the file unchanged.

Update `mod_auth_cas` settings

Edit the file `/etc/httpd/conf.d/cas.conf` on the client server (***casdev-casapp***) and create another `<Directory>` element for the new secure content area created above:

```
<Directory "/var/www/html/secured-by-cas-duo">
    <IfModule mod_auth_cas.c>
        AuthType        CAS
        CASAuthNHeader  On
    </IfModule>

    Require valid-user
</Directory>
```

Except for the path name of the directory, it should be identical to the other `<Directory>` elements.

Update the public content page

Update `/var/www/html/index.php` to include a link to the new secure area:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1">
    <link rel="stylesheet"
      href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.m
in.css">
  </head>
  <body>
    <div class="container">
      <h1>Hello, World!</h1>
        <p><big>The quick brown fox jumped over the lazy dogs.</bi
g></p>
        <p><big>Click <a href="secured-by-cas/index.php">here</a> fo
r some
          content secured by username and password.</big></p>
        <p><big>Click <a href="return-mapped/index.php">here</a> to s
ee the
          results of the "Return Mapped" attribute release policy.</b
ig></p>
        <p><big>Click <a href="secured-by-cas-duo/index.php">her
e</a> for some
          content secured by username/password and Duo MFA.</big></p>
    </div>
  </body>
</html>
```

Restart HTTPD

Run the command

```
casdev-casapp# systemctl restart httpd
```

to restart the HTTPD server with the new configuration. Check the log files in `/var/log/httpd` for errors.

Update the service registry

Although it's possible to enable MFA across the board for all services by setting properties in `cas.properties` (see CAS 5: Configuration Properties: Multi-factor Authentication), it's usually preferable to configure it on a per-service basis in the service registry.

Create a second service definition for the CAS client

Make a copy of `etc/cas/services/ApacheSecuredByCAS-1504122840.json` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and call it `ApacheSecuredByCASandDuo-1504200420.json` (replace `1504200420` with the current `date +%s` or `YYYYMMDDhhmmss` value):

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# cp -p etc/cas/services/ApacheSecuredByCAS-1504122840.j
son etc/cas/services/ApacheSecuredByCASandDuo-201700831132700.json
```

Then edit `etc/cas/services/ApacheSecuredByCASandDuo-1504200420.json` and do the following:

1. Change the `serviceId` property to reflect the path to the secure area created in the previous step (page 169).
2. Change the `id` property to a unique value (make sure this value matches the one in the filename).
3. Change the `description` property to include the Duo MFA requirement.
4. Add the `multifactorPolicy` property as shown below.
5. Change the `evaluationOrder` property to a different value.

When done, the file should look something like this:

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^https://casdev-casapp.newschool.edu/secured-by-ca
s-duo(\\z|/.*)",
  "name" : "Apache Secured By CAS and Duo",
  "id" : 1504200420,
  "description" : "CAS development Apache mod_auth_cas server with us
ername/password and Duo MFA protection",
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePoli
cy"
  },
  "multifactorPolicy" : {
    "@class" : "org.apereo.cas.services.DefaultRegisteredServiceMulti
factorPolicy",
    "multifactorAuthenticationProviders" : [ "java.util.LinkedHashSe
t", [ "mfa-duo" ] ]
  },
  "evaluationOrder" : 1200
}
```

The `multifactorPolicy` added here defines a single MFA provider, `mfa-duo`. It does not allow the MFA requirement to be bypassed (meaning that users not registered with Duo will not be able to log in), and it will fail “closed,” meaning that if for some reason the Duo service is unavailable, users will not be able to log in.

References

- CAS 5: Duo Security Authentication
- CAS 5: Configuration Properties: Multi-factor Authentication

Install and test the application

Before multi-factor authentication can be used, the rebuilt CAS application and the updated configuration files must be installed. Then everything can be tested by accessing the “public” part of the client application web site and then clicking on the link to the “secure” section. At that point, the browser should be redirected to the CAS server, where upon successful authentication the secure content, including the values of the attributes, will be displayed.

Because both the load balancer and the CAS server use cookies, it’s usually best to perform testing with an “incognito” or “private browsing” instance of the web browser that deletes all cookies each time it is closed.

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS application and configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Shut down all but one of the pool servers

Operating CAS with a pool of servers instead of a single server requires special configuration. Because that configuration hasn't been completed yet, testing must be performed against a single server. Therefore, the other servers in the pool should be shut down so that the load balancer will direct all traffic to that single server. Run the command

```
# systemctl stop tomcat
```

on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

Access the public site

Open up a web browser (in "incognito" or "private browsing" mode) and enter the URL of the CAS-enabled web site:

```
https://casdev-casapp.newschool.edu/
```

The contents of `/var/www/html/index.php` should be displayed, looking something like this:

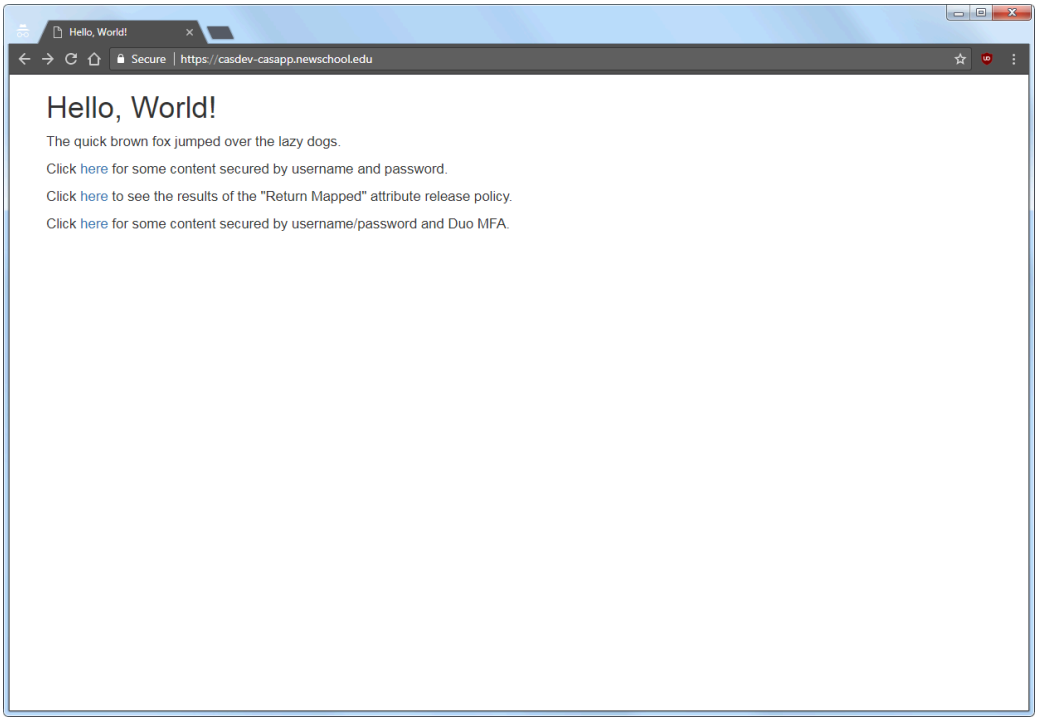

Figure 14. The "public" site

Access the secure area

Click on the second “here” link to access the content secured by CAS and Duo, and you will be redirected to the CAS server login page, as shown below:

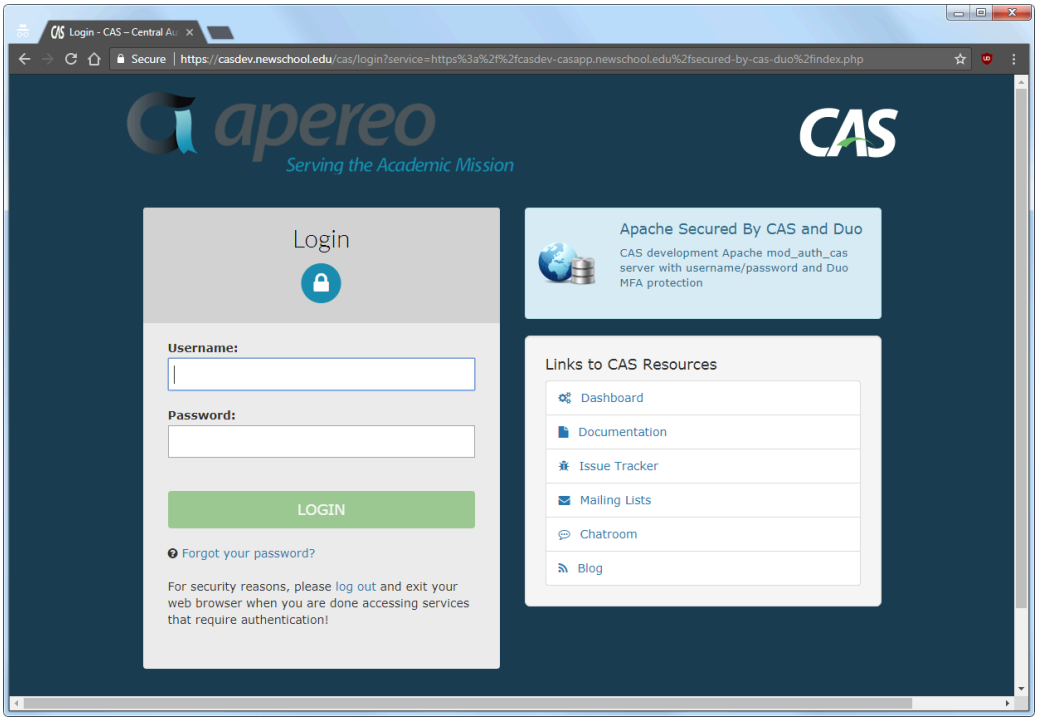

Figure 15. The CAS login page

Note that the contents of the `name` field from the service registry are displayed at the top of the right-hand column; make sure that `Apache Secured by CAS and Duo` is displayed here. Enter a valid username and password (Active Directory or LDAP) that is also registered with Duo and, upon successful first-stage authentication, the Duo MFA authentication page will appear, as shown below:

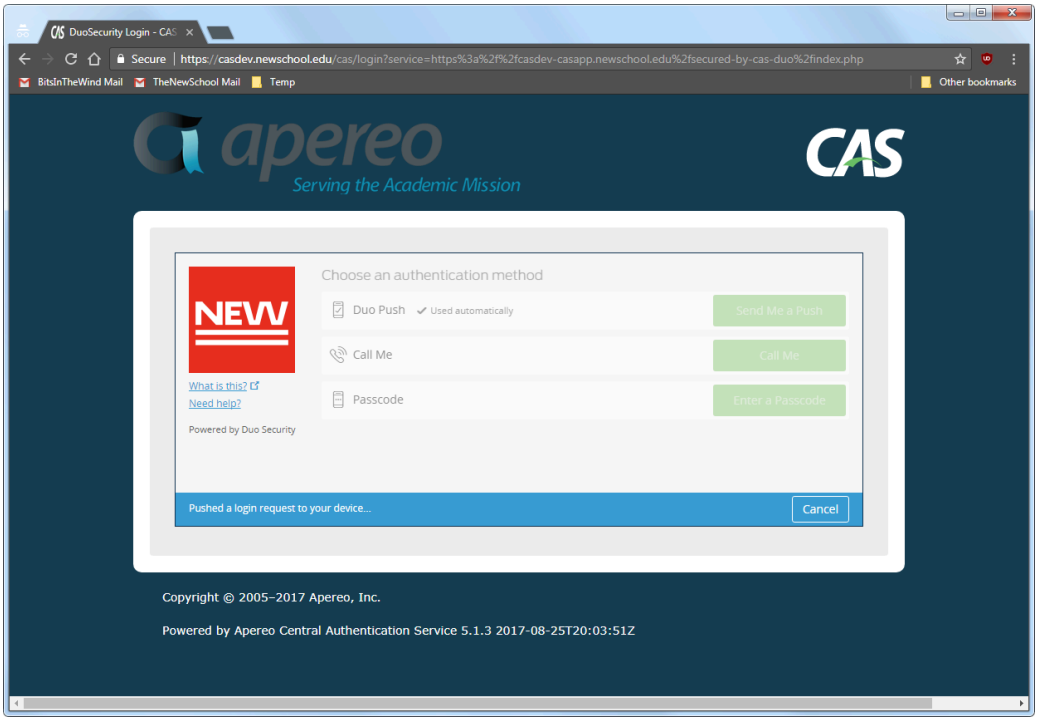

Figure 16. The Duo authentication page

Upon successful Duo authentication, the contents of `/var/www/html/secured-by-cas-duo/index.php` will be displayed:

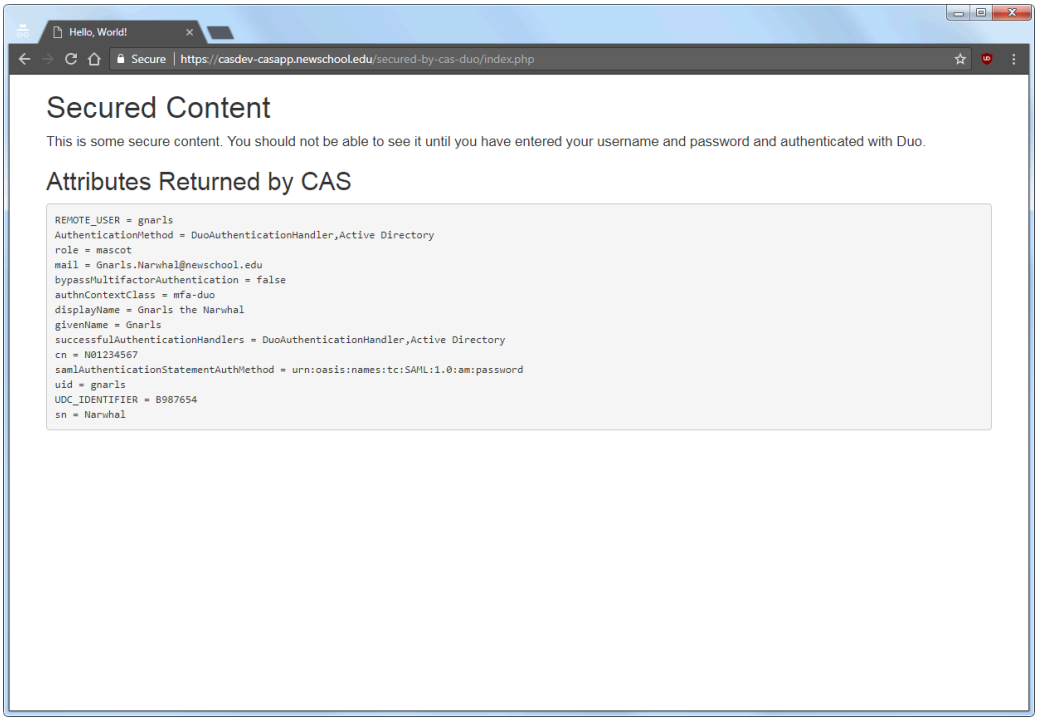

Figure 17. The "secure" content

Restart the pool servers

One testing is complete, run the command

```
# systemctl start tomcat
```

on each of the pool servers shut down previously.

Commit changes to Git

Before moving on to the next task, commit the changes made to `pom.xml` and `cas.properties`, as well as the new `etc/cas/services/ApacheSecuredByCASandDuo-1504200420.json` file, to Git to make them easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git add etc/cas/services/ApacheSecuredByCASandDuo-1504
200420.json
casdev-master# git add pom.xml
casdev-master# git commit -m "Added Duo MFA support"
[newschool-casdev 7a51280] Added Duo MFA support
 3 files changed, 38 insertions(+)
 create mode 100644 etc/cas/services/ApacheSecuredByCASandDuo-1504200
420.json
casdev-master#
```

on the master build server (***casdev-master***). The `git commit` command will not bring up a text editor as it did last time, since we provided the commit message on the command line.

Adding SAML support

Summary: CAS 5's native capability to operate as a SAML2 Identity Provider will be used to provide authentication and single sign-on support to services that do not support the CAS protocol.

The CAS protocol is our preferred protocol for authentication and single sign-on: it's easy to understand, easy to configure, and "just works." Unfortunately, while CAS is pretty well supported by applications and services designed for the higher education market, is is much less widely supported by applications and services targeted mainly at the corporate world. These applications and services typically use the SAML2 protocol instead. Prior to CAS 5, supporting both protocols together required setting up a Shibboleth server alongside the CAS server, and configuring one server to use the other as its authentication source. Although this more or less worked, it was difficult to manage, and didn't handle all aspects of single sign-on (especially single log-out) very cleanly.

It's important to understand two terms when talking about SAML, *Identity Provider* and *Service Provider*:

Identity Provider (IdP)	A SAML Identity Provider (IdP) is a service that authenticates users ("principals") by means such as usernames, passwords, and multi-factor authentication schemes. An authenticated user is given a security token that he or she can present to service providers (SPs, see below) to gain access to their services. The IdP also accepts users' security tokens from SPs and returns an indication of whether or not they are valid. CAS 5 has added full native SAML2 support, enabling the CAS server to function as an IdP and eliminating our dependency on Shibboleth.
Service Provider (SP)	A SAML Service Provider (SP) is an entity that provides web services to users. When a user attempts to access the service, he or she must present the service with a security token generated by a recognized IdP. The SP validates the token with the IdP. If the user does not have a token, or the presented token is invalid, the SP sends the user to the IdP to obtain a new one. The SAML client (page 199) that we will build in the next section will be our SP for testing purposes.

Add the SAML2 IdP dependency to the project object model

To add SAML2 IdP support to the CAS server, edit the file `pom.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the dependencies section (around line 69), which should look something like this:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
         <groupId>org.apereo.cas</groupId>
         <artifactId>cas-server-support-duo</artifactId>
         <version>${cas.version}</version>
    </dependency>
</dependencies>
```

Insert the new dependency for the SAML2 IdP support just after the `cas-server-support-saml` dependency added previously:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml-idp</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
         <groupId>org.apereo.cas</groupId>
         <artifactId>cas-server-support-duo</artifactId>
         <version>${cas.version}</version>
    </dependency>
</dependencies>
```

This will instruct Maven to download the appropriate code modules and build them into the server.

ℹ **Note:** There are two different SAML dependencies: `cas-server-support-saml` enables support for returning user attributes to client applications; `cas-server-support-saml-idp` enables support for using the CAS server as a SAML2 Identity Provider. Both dependencies should be included in `pom.xml`.

Rebuild the server

Run Maven to rebuild the server according to the new model:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 01:00 min
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 35M/84M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

References

- CAS 5: Configuring SAML2 Authentication

Update the server configuration

The CAS server's IdP functionality requires some adjustments to Tomcat's settings, a couple of new CAS property settings, and the creation of a cache directory where SAML2 metadata can be stored.

Adjust Tomcat settings

The SAML2 protocol requires the use of large (2MB) HTTP header sets and large (2MB) HTTP POST payloads. To enable this support on Tomcat's HTTP connector, edit the file `/opt/tomcat/latest/conf/server.xml` on the master build server (***casdev-master***) and locate the definition of the HTTPS connector (around line 89), which should look something like this:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioPr
otocol"
    sslImplementationName="org.apache.tomcat.util.net.openssl.OpenSSL
Implementation"
    SSLEnabled="true" connectionTimeout="20000" maxThreads="150">
    <SSLHostConfig
        ciphers="ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POL
Y1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDH
E-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES12
8-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDH
E-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:EC
DHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:E
CDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RS
A-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RS
A-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA3
84:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DS
S"
        honorCipherOrder="true" protocols="all,-SSLv2Hello,-SSLv2,-SS
Lv3"
        disableSessionTickets="true">
        <Certificate
            certificateKeystoreFile="/opt/tomcat/keystore.jks"
            certificateKeystorePassword="changeit"
            type="RSA" />
    </SSLHostConfig>
    <UpgradeProtocol className="org.apache.coyote.http2.Http2Protoco
l" />
</Connector>
```

Add the `maxHttpHeaderSize` and `maxPostSize` attributes to the connector definition, like this:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
    sslImplementationName="org.apache.tomcat.util.net.openssl.OpenSSLImplementation"
    SSLEnabled="true" maxHttpHeaderSize="2097152" maxPostSize="2097152"
    connectionTimeout="20000" maxThreads="150">
```

Configure SAML IdP properties

Add the following settings to `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) to configure the SAML IdP:

```
cas.authn.samlIdp.entityId:             ${cas.server.prefix}/idp
cas.authn.samlIdp.scope:                newschool.edu
```

The `entityId` parameter is the URL by which the IdP is known to clients (SPs). The `scope` parameter identifies the “scope” in which attributes returned by the IdP apply; this is typically a DNS domain.

Create the metadata cache directory

Create the directory `etc/cas/saml` in the `cas-overlay-template` directory on the master build server:

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# mkdir etc/cas/saml
```

Adjust the server installation script

If you created an installation shell script (page 99) earlier, edit that script and, just after the line that extracts the `tar` file (around line 12), add a `chmod` command to restore group write permission to the `etc/cas/saml` directory so the CAS server can create files there:

```
cd /
rm -rf etc/cas/config etc/cas/services
tar xzf /tmp/cassrv-files.tgz etc/cas
chmod g+w etc/cas/saml
```

References

- CAS 5: Configuration Properties: SAML IdP

Update the service registry

When the CAS server starts and initializes the SAML IdP, it creates a new (undocumented, as of this writing) endpoint, `${cas.server.prefix}/idp/profile/SAML2/Callback`. It then checks the service registry to see if there is an existing service definition whose `serviceId` will match then endpoint and allow access. If there isn't one, the CAS server will create a new service definition for the endpoint, and save it to the service registry. If the CAS server does not have permission to create new entries in the service registry for whatever reason, then the save will fail, and the server will not start. The error messages in the CAS log file will be somewhat cryptic in this case, because they won't refer to SAML or the IdP at all. But the relevant lines will look something like this:

```
============================================================
WHO: audit:unknown
WHAT: IO error opening file stream.
ACTION: SAVE_SERVICE_FAILED
APPLICATION: CAS
WHEN: Ddd Mon DD hh:mm:ss zzz YYYY
CLIENT IP ADDRESS: unknown
SERVER IP ADDRESS: unknown
============================================================

(...lots of stack trace output...)

Caused by: java.io.FileNotFoundException: /etc/cas/services/RegexRegi
steredService-6805904835673174978.json (Permission denied)
```

As luck would have it, back when we first set up our service registry (page 107), we created a "wildcard" service definition (`HTTPSandIMAPSwildcard-1503925297.json`) that will match the endpoint above, so the CAS server will not create a new service definition and try to save it in the registry. However, while a wildcard service definition is fine in a development or test environment, we won't have such a thing in our production environment.

Create a service definition for the IdP endpoint

To avoid any risk of the server failing to start as described above, and also to make it clear that the IdP endpoint is a "desired" service, we will explicitly create a service definition file for it.

Make a copy of `etc/cas/services/HTTPSandIMAPSwildcard-1503925297.json` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and call it `SAML2CallbackProfile-1509029745.json` (replace `1509029745` with the current `date +%s` or `YYYYMMDDhhmmss` value):

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# cp -p etc/cas/services/HTTPSandIMAPSwildcard-150392529
7.json etc/cas/services/SAML2CallbackProfile-1509029745.json
```

Then edit `etc/cas/services/SAML2CallbackProfile-1509029745.json` and do the following:

1. Change the `serviceId` property to `https://casdev.newschool.edu/cas/idp/profile/SAML2/Callback.+` (note the `.+` regular expression component on the end).
2. Change the `name` property to `SAML Authentication Request`.
3. Change the `id` property to a unique value (make sure this value matches the one in the filename).
4. Change the `evaluationOrder` property to a value smaller than the other services' values, to (hopefully) ensure that this definition will always match the endpoint.
5. Add a comment to explain what the definition is for (optional).

When done, the file should look something like this:

```
{
  /*
   * The CAS SAML IdP creates this endpoint as part of its initializa
tion
   * process at server startup time. If the service registry doesn't
already
   * contain an entry whose serviceId matches the endpoint, CAS will
create
   * a new service definition and save it to the registry. If the CA
S server
   * doesn't have write access to the registry, then the save will fa
il and
   * the server will not start.
   *
   * To avoid that situation, and to make it clear that this endpoin
t is a
   * "desired" service, it is defined explicitly here.
   */
  "@class" :            "org.apereo.cas.services.RegexRegisteredServi
ce",
  "serviceId" :         "https://casdev.newschool.edu/cas/idp/profil
e/SAML2/Callback.+",
  "name" :              "SAML Authentication Request",
  "id" :                1509029745,
  "evaluationOrder" :   100
}
```

Install and test the IdP

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS application and configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Check that the IdP is working on the master build server

To check that the IdP functionality is working, use `curl` to request the IdP's metadata:

```
casdev-master# curl -k https://casdev-master.newschool.edu:8443/cas/idp/metadata
<?xml version="1.0" encoding="UTF-8"?>
<EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:shibmd="urn:mace:shibboleth:metadata:1.0" xmlns:xml="http://www.w3.org/XML/1998/namespace" xmlns:mdui="urn:oasis:names:tc:SAML:metadata:ui" entityID="https://casdev.newschool.edu/idp">
    <IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol urn:oasis:names:tc:SAML:1.1:protocol urn:mace:shibboleth:1.0">
        <Extensions>
            <shibmd:Scope regexp="false">newschool.edu</shibmd:Scope>
        </Extensions>
        <KeyDescriptor use="signing">
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate>MIIDMjCCAhqgAwIBAgIVAJ52X1Nr07ZSRcD/sf+/sru29lnuMA0GCSqGSIb3DQEB
CwUAMB8xHTAbBgNVBAMMFGNhc2Rldi5uZXdzY2hvb2wuZWR1MB4XDTE3MTAyMzE4MzQxNloXDTM3MTAyMzE4MzQxNlowHzEdMBsGA1UEAwwUY2FzZGV2Lm5ld3NjaG9vbC5lZHUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC9Ewa1ETMNN0rFmpzNZln2x638hQjX78T5F78nk0/K37aHXaOUH7AgQshFuDVct+QfApur4FRtJl/HF+LMJRSastrZDnjFEBMkb4iVsXZVQ+H+UsPEyVmPYfzTjtNwJFLsQfNStNmHF7AGy+/3WxTa0KZKXq2yzGKIy9cnvmqzvve2PVmBOhwn3zFiT0ZgY+RoWzQVtIzkFst/OATUg2c93LucM0x6Ec7VAYrH6sgBNjiP3NfsXv49mJhTJXuAkeWo3wGApR6R4ezmmeUm0HrNCZQ6dTRDGl5qZmax0ltCS2iJnwtd8BUSvwzZT1YrX57Qg2jrMZoo+SHRFpCrQu79AgMBAAGjZTBjMB0GA1UdDgQWBBSDv5Ny5w80rMMx/VwYv5lsTVqYqzBCBgNVHREEOzA5ghRjYXNkZXYubmV3c2Nob29sLmVkdYYhY2FzZGV2Lm5ld3NjaG9vbC5lZHUvaWRwL21ldGFkYXRhMA0GCSqGSIb3DQEBCwUAA4IBAQAmhBGB9wOMJhZaK9g1cX8bk9AcVeJAbltx/MUjiyopj98zy0DuuhUVPT1F2c36FFYzMx5JrCOfIwOlBbDHSiSCTUapjQe2+McvTH2/Cvi2lNJ4rjlZpIdINJdzwEcTUnJYnUnWoO0oPALM/V/IHlzqBR+lOtcv1wW19+WYT+BLzNMKc/ThmK2kZK88dpyLCyrxfiv1Om3dYNGMoM6d+kQLuxM+h3qhnFVo/Xg3GJEImXoAYPS8A165y8kTJOa6FuvTR9kjmLoZ5E+936HVMrpjJZZO/XduOZ7k5qN20wRYMMyA7Lf85hgKItEmTJe/McrQgthbRaSsgY5qnrndd+52</ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </KeyDescriptor>
        <KeyDescriptor use="encryption">
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate>MIIDMjCCAhqgAwIBAgIVAMX1YMhKcxKp/uraKDAmWA6u1AjnMA0GCSqGSIb3DQEB
CwUAMB8xHTAbBgNVBAMMFGNhc2Rldi5uZXdzY2hvb2wuZWR1MB4XDTE3MTAyMzE4MzQxN1oXDTM3MTAyMzE4MzQxN1owHzEdMBsGA1UEAwwUY2FzZGV2Lm5ld3NjaG9vbC5lZHUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCGo/8pOKu2ECEQ
```

```
XxHedB1RVTa4DGM5BYsubhrp6ANSS9eym2pGUi6hMJVPgdTaI3pnPecC8sk9DnQ5
O/7C7/utdit86bvDc4hUTJ1+W1OY2GrfS1QGqU2PFE0CtiJWtQp+g8ypBbXzcP6W
Zz3DRJMe1Jt44/fNoCgRHeiNVBWs3v2oNlpmA8A2+g9RKA9slrS4YuTkXNoTUYgo
JuZSDNhu0sZFx1WYElOrWs7ry7NCO1lJrl38UOGt9RFsF/7XRXSx4df2Hkr+XrIz
sVigUlxt1PiL8F03nH3tPMh4eIinmZ3F0r0vrFhrsc7Rzwkty4Bu/xEkA4mDE85k
YHy2UwjrAgMBAAGjZTBjMB0GA1UdDgQWBBSaNMHTF540kZ9XPGUuLb5EIM409TBC
BgNVHREEOzA5ghRjYXNkZXYubmV3c2Nob29sLmVkdYYhY2FzZGV2Lm5ld3NjaG9v
bC5lZHUvaWRwL21ldGFkYXRhMA0GCSqGSIb3DQEBCwUAA4IBAQAzHson1VU4/ZdC
pDXIUMACmN+WZPrgGQ+Xd3w179X1VrYmmU2KrWTu3Wq+svn0aUbEcWY+dDs7wFK2
gYXAYI8p4RY37n+Z/m8MyipZDHqZ6bloNsfjJ7j+L7yxLC5AQKgUpUON1kvsainI
zwr/W4Q0udFQzX1qxI6b9EF23V+8r4Hkl105Xh3bgkX5c52qUy71BoBDNvxvWXfT
q6NiXFEIX821Jc57xjqOWyD7tdfLnY5Imkb1ZQ5GlHHlWR6UKuZojIuBwT9Fotha
dgmL7gvBapJdfV1aNaz1MfC6REKnE5P+SZboEBGWpe8pGFSBKvyNAnfzCKNycSzH
/k3s3wIp</ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </KeyDescriptor>

        <!--
        <ArtifactResolutionService Binding="urn:oasis:names:tc:SAM
L:1.0:bindings:SOAP-binding"
                                   Location="https://casdev.newschoo
l.edu/cas/idp/profile/SAML1/SOAP/ArtifactResolution" index="1"/>
        -->

        <SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bin
dings:HTTP-POST" Location="https://casdev.newschool.edu/cas/idp/profi
le/SAML2/POST/SLO"/>

        <NameIDFormat>urn:mace:shibboleth:1.0:nameIdentifier</NameIDF
ormat>
        <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:trans
ient</NameIDFormat>

        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bin
dings:HTTP-POST" Location="https://casdev.newschool.edu/cas/idp/profi
le/SAML2/POST/SSO"/>
        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bin
dings:HTTP-POST-SimpleSign" Location="https://casdev.newschool.edu/ca
s/idp/profile/SAML2/POST-SimpleSign/SSO"/>
        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bin
dings:HTTP-Redirect" Location="https://casdev.newschool.edu/cas/idp/p
rofile/SAML2/Redirect/SSO"/>
        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bin
dings:SOAP" Location="https://casdev.newschool.edu/cas/idp/profile/SA
ML2/SOAP/ECP"/>
    </IDPSSODescriptor>
```

```
    <!--
    <AttributeAuthorityDescriptor protocolSupportEnumeration="urn:oas
is:names:tc:SAML:1.1:protocol urn:oasis:names:tc:SAML:2.0:protocol">
        <Extensions>
            <shibmd:Scope regexp="false">newschool.edu</shibmd:Scope>
        </Extensions>
        <KeyDescriptor use="signing">
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate>MIIDMjCCAhqgAwIBAgIVAJ52X1Nr0
7ZSRcD/sf+/sru29lnuMA0GCSqGSIb3DQEB
CwUAMB8xHTAbBgNVBAMMFGNhc2Rldi5uZXdzY2hvb2wuZWR1MB4XDTE3MTAyMzE4
MzQxNloXDTM3MTAyMzE4MzQxNlowHzEdMBsGA1UEAwwUY2FzZGV2Lm5ld3NjaG9v
bC5lZHUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC9Ewa1ETMNN0rF
mpzNZln2x638hQjX78T5F78nk0/K37aHXaOUH7AgQshFuDVct+QfApur4FRtJl/H
F+LMJRSastrZDnjFEBMkb4iVsXZVQ+H+UsPEyVmPYfzTjtNwJFLsQfNStNmHF7AG
y+/3WxTa0KZKXq2yzGKIy9cnvmqzvve2PVmBOhwn3zFiT0ZgY+RoWzQVtIzkFst/
OATUg2c93LucM0x6Ec7VAYrH6sgBNjiP3NfsXv49mJhTJXuAkeWo3wGApR6R4ezm
meUm0HrNCZQ6dTRDGl5qZmax0ltCS2iJnwtd8BUSvwzZT1YrX57Qg2jrMZoo+SHR
FpCrQu79AgMBAAGjZTBjMB0GA1UdDgQWBBSDv5Ny5w80rMMx/VwYv5lsTVqYqzBC
BgNVHREEOzA5ghRjYXNkZXYubmV3c2Nob29sLmVkdYYhY2FzZGV2Lm5ld3NjaG9v
bC5lZHUvaWRwL21ldGFkYXRhMA0GCSqGSIb3DQEBCwUAA4IBAQAmhBGB9wOMJhZa
K9g1cX8bk9AcVeJAbltx/MUjiyopj98zy0DuuhUVPT1F2c36FFYzMx5JrCOfIwOl
BbDHSiSCTUapjQe2+McvTH2/Cvi2lNJ4rjlZpIdINJdzwEcTUnJYnUnWoO0oPALM
/V/IHlzqBR+lOtcv1wW19+WYT+BLzNMKc/ThmK2kZK88dpyLCyrxfiv1Om3dYNGM
oM6d+kQLuxM+h3qhnFVo/Xg3GJEImXoAYPS8A165y8kTJOa6FuvTR9kjmLoZ5E+9
36HVMrpjJZZO/XduOZ7k5qN20wRYMMyA7Lf85hgKItEmTJe/McrQgthbRaSsgY5q
nrndd+52</ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </KeyDescriptor>
        <AttributeService Binding="urn:oasis:names:tc:SAML:1.0:bindin
gs:SOAP-binding" Location="https://casdev.newschool.edu/cas/idp/profi
le/SAML1/SOAP/AttributeQuery"/>
        <AttributeService Binding="urn:oasis:names:tc:SAML:2.0:bindin
gs:SOAP" Location="https://casdev.newschool.edu/cas/idp/profile/SAML
2/SOAP/AttributeQuery"/>
    </AttributeAuthorityDescriptor>
    -->

    <!--
    <Organization>
        <OrganizationName xml:lang="en">Institution Name</Organizatio
nName>
        <OrganizationDisplayName xml:lang="en">Institution DisplayNam
e</OrganizationDisplayName>
```

```
        <OrganizationURL xml:lang="en">URL</OrganizationURL>
    </Organization>
    <ContactPerson contactType="administrative">
        <GivenName>John Smith</GivenName>
        <EmailAddress>jsmith@example.org</EmailAddress>
    </ContactPerson>
    <ContactPerson contactType="technical">
        <GivenName>John Smith</GivenName>
        <EmailAddress>jsmith@example.org</EmailAddress>
    </ContactPerson>
    <ContactPerson contactType="support">
        <GivenName>IT Services Support</GivenName>
        <EmailAddress>support@example.org</EmailAddress>
    </ContactPerson>
    -->
</EntityDescriptor>
casdev-master#
```

You should receive a relatively lengthy XML document back.

> **Note:** The `-k` option to `curl` tells it not to verify the SSL certificate of the server; this is necessary because the server is identifying itself as ***casdev.newschool.edu***, not ***casdev-master.newschool.edu***.

Copy CAS-generated IdP metadata to the overlay template

When the CAS server was started with IdP support for the first time (above), it generated IdP-specific signing and encryption keys and certificates to be used when communicating with SAML2 clients. The server wrote copies of these keys and certificates, as well as the IdP metadata, to files in `/etc/cas/saml`:

```
casdev-master# ls -asl /etc/cas/saml
total 28
4 drwxrwx---. 2 root   tomcat 4096 Mmm dd hh:mm .
0 drwxr-x---. 5 root   tomcat   45 Mmm dd hh:mm ..
4 -rw-rw----. 1 tomcat tomcat 1168 Mmm dd hh:mm idp-encryption.crt
4 -rw-rw----. 1 tomcat tomcat 1675 Mmm dd hh:mm idp-encryption.key
8 -rw-rw----. 1 tomcat tomcat 7383 Mmm dd hh:mm idp-metadata.xml
4 -rw-rw----. 1 tomcat tomcat 1168 Mmm dd hh:mm idp-signing.crt
4 -rw-rw----. 1 tomcat tomcat 1679 Mmm dd hh:mm idp-signing.key
casdev-master#
```

To make sure that these files are the same across all the CAS servers (so that any back-end server can talk to any client), copy them into the overlay template's `etc/cas` directory:

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# cp /etc/cas/saml/idp-* etc/cas/saml
```

Install on the CAS servers

Once everything seems to be running correctly on the master build server (further testing will have to wait until we build the SAML client (page 199)), copy the updated Tomcat settings to the CAS servers:

```
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /etc/tomcat/server.xml casdev-${host}:/etc/tomcat/server.xml
> done
casdev-master#
```

Then copy the new CAS application files to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Check that the IdP is working on the CAS servers

Use the `curl` command again to check that the IdP is working on the CAS servers. This time the `-k` option is not necessary:

```
casdev-master# curl https://casdev.newschool.edu/cas/idp/metadata
(XML output)
casdev-master#
```

Commit changes to Git

Before moving on to building the SAML client, commit the changes made to `pom.xml`, as well as the new `etc/cas/saml` directory and the IdP-specific service definition, to Git to make them easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/saml
casdev-master# git add pom.xml
casdev-master# git add etc/cas/services/SAML2CallbackProfile-1509029745.json
casdev-master# git commit -m "Added SAML IdP support"
[newschool-casdev 3bf73e3] Added SAML IdP support
 6 files changed, 224 insertions(+)
 create mode 100644 etc/cas/saml/idp-encryption.crt
 create mode 100644 etc/cas/saml/idp-encryption.key
 create mode 100644 etc/cas/saml/idp-metadata.xml
 create mode 100644 etc/cas/saml/idp-signing.crt
 create mode 100644 etc/cas/saml/idp-signing.key
 create mode 100644 etc/cas/services/SAML2CallbackProfile-1509029745.json
casdev-master#
```

on the master build server (***casdev-master***).

Building the SAML client

Summary: To facilitate development and testing, a client application that interacts with the CAS server via SAML is needed.

Now that SAML2 Identity Provider (IdP) support has been added to the CAS server (page 181), we can build a Service Provider (SP) client application to talk to it.

Our SP will be an Apache HTTPD web server that offers both public content that anyone can access, and “secure” content that can only be accessed by authenticated and authorized users. The IdP functionality of the CAS server will be used to perform those authentication and authorization decisions.

Install the Shibboleth SP

The Shibboleth Service Provider (SP) allows an Apache web server to interact with a SAML Identity Provider (IdP) via the SAML2 protocol. The SP is comprised of an Apache HTTPD module (`mod_shib`) and a system daemon (`shibd`) that handles state management and most of the actual SAML processing (the module communicates with the daemon; the daemon communicates with the IdP). Red Hat does not offer this software, but the Shibboleth Consortium uses the OpenSUSE Build Service to distribute its packages, so we can still install it with `yum` .

> ℹ **Note:** The steps in this section should be performed on the client server (***casdev-samlsp***), not the master build server (***casdev-master***).

Add the Shibboleth repository to `yum`

Before we can use `yum` to install the Shibboleth SP, we have to teach it about the Shibboleth repository from the OpenSUSE Build Service. Run the command

```
casdev-samlsp# curl http://download.opensuse.org/repositories/securit
y:/shibboleth/CentOS_7/security:shibboleth.repo -o /etc/yum.repos.d/s
ecurity\:shibboleth.repo
  % Total    % Received % Xferd  Average Speed   Time    Time     Tim
e  Current
                                 Dload  Upload   Total   Spent    Lef
t  Speed
100   266  100   266    0     0   1225      0 --:--:-- --:--:--
--:--:--  1231
casdev-samlsp#
```

to download the repo configuration file.

> ⚠ **Important:** From the Shibboleth Wiki : “A special note applies to Red Hat 7 and probably all future versions: because of Red Hat’s licensing restrictions, it’s now impossible for the build service to target Red Hat 7 directly. However, CentOS is an identical system, and the packages for it work on the equivalent Red Hat versions, so Red Hat 7 deployments should rely on the CentOS 7 package repository.”

Install the Shibboleth SP

Install the Shibboleth SP and its dependencies by running the commands

```
casdev-samlsp# yum -y install shibboleth
```

Create a TLS/SSL certificate for the SP

The SP uses its own TLS/SSL certificate for signing and encrypting communications with the IdP. Run the commands

```
casdev-samlsp# cd /etc/shibboleth
casdev-samlsp# ./keygen.sh -h casdev-samlsp.newschool.edu -e http
s://casdev.newschool.edu/shibboleth -f -u shibd -y 10
Generating a 3072 bit RSA private key
.........++
.................................................++
writing new private key to './sp-key.pem'
-----
casdev-samlsp#
```

Any error messages about being unable to remove `./sp-key.pem` or `./sp-cert.pem` may be safely ignored.

Configure `systemd` to start `shibd`

RHEL 7 uses `systemd` to manage system resources. Run the command

```
casdev-samlsp# systemctl enable shibd
```

to enable the `shibd` service in `systemd`. This will cause `systemd` to start `shibd` at system boot time. Additionally, the following commands may now be used to manually start, stop, restart, and check the status of the `shibd` service:

```
# systemctl start shibd
# systemctl stop shibd
# systemctl restart shibd
# systemctl status shibd
```

Test that HTTPD and `shibd` can communicate

Before configuring the SP further, check that Apache HTTP and `shibd` are able to communicate. Run the commands

```
casdev-samlsp# systemctl start shibd
casdev-samlsp# systemctl restart HTTPD
```

to start the `shibd` daemon and restart HTTPD to load the `mod_shib` module. Then use `curl` to retrieve the SP's status page:

```
casdev-samlsp# curl -k https://127.0.0.1/Shibboleth.sso/Status
<StatusHandler time='2017-10-25T19:12:42Z'><Version Xerces-C='3.1.1'
XML-Tooling-C='1.6.0' XML-Security-C='1.7.3' OpenSAML-C='2.6.0' Shibb
oleth='2.6.0'/><NonWindows sysname='Linux' nodename='casdev-samlsp.ne
wschool.edu' release='3.10.0-693.2.2.el7.x86_64' version='#1 SMP Sat
Sep 9 03:55:24 EDT 2017' machine='x86_64'/><SessionCache><OK/></Sessi
onCache><Application id='default' entityID='https://sp.example.org/sh
ibboleth'/><Handlers><Handler type='ArtifactResolutionService' Locati
on='/Artifact/SOAP' Binding='urn:oasis:names:tc:SAML:2.0:bindings:SOA
P'/><Handler type='AssertionConsumerService' Location='/SAML2/POST' B
inding='urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST'/><Handler typ
e='AssertionConsumerService' Location='/SAML2/POST-SimpleSign' Bindin
g='urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST-SimpleSign'/><Handl
er type='AssertionConsumerService' Location='/SAML2/Artifact' Bindin
g='urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact'/><Handler typ
e='AssertionConsumerService' Location='/SAML2/ECP' Binding='urn:oasi
s:names:tc:SAML:2.0:bindings:PAOS'/><Handler type='AssertionConsumerS
ervice' Location='/SAML/POST' Binding='urn:oasis:names:tc:SAML:1.0:pr
ofiles:browser-post'/><Handler type='AssertionConsumerService' Locati
on='/SAML/Artifact' Binding='urn:oasis:names:tc:SAML:1.0:profiles:art
ifact-01'/><Handler type='SessionInitiator' Location='/Login'/><Handl
er type='SingleLogoutService' Location='/SLO/SOAP' Binding='urn:oasi
s:names:tc:SAML:2.0:bindings:SOAP'/><Handler type='SingleLogoutServic
e' Location='/SLO/Redirect' Binding='urn:oasis:names:tc:SAML:2.0:bind
ings:HTTP-Redirect'/><Handler type='SingleLogoutService' Location='/S
LO/POST' Binding='urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST'/><H
andler type='SingleLogoutService' Location='/SLO/Artifact' Binding='u
rn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact'/><Handler type='Lo
goutInitiator' Location='/Logout'/><Handler type='MetadataGenerator'
Location='/Metadata'/><Handler type='Status' Location='/Status'/><Han
dler type='Session' Location='/Session'/><Handler type='DiscoveryFee
d' Location='/DiscoFeed'/></Handlers><md:KeyDescriptor xmlns:md="ur
n:oasis:names:tc:SAML:2.0:metadata" use="signing"><ds:KeyInfo xmlns:d
s="http://www.w3.org/2000/09/xmldsig#"><ds:KeyName>casdev-samlsp.news
chool.edu</ds:KeyName><ds:KeyName>https://casdev.newschool.edu/shibbo
leth</ds:KeyName><ds:X509Data><ds:X509SubjectName>CN=casdev-samlsp.ne
wschool.edu</ds:X509SubjectName><ds:X509Certificate>MIIEQTCCAqmgAwIBA
gIJAJgyfve+2p4MMA0GCSqGSIb3DQEBCwUAMCYxJDAiBgNV
BAMTG2Nhc2Rldi1zYW1sc3AubmV3c2Nob29sLmVkdTAeFw0xNzEwMjUxOTA5NTNa
Fw0yNzEwMjMxOTA5NTNaMCYxJDAiBgNVBAMTG2Nhc2Rldi1zYW1sc3AubmV3c2No
b29sLmVkdTCCAaIwDQYJKoZIhvcNAQEBBQADggGPADCCAYoCggGBAKnAVWGK7S1f
nmL41kvXlU9l2BDQNKEpoEnu428Bg8Az/5e1wxs2eMk9XymPobEJ5LlT0Fyr4nSl
NqRSHkCOFdoll+W8qfqn3AxgYaGDccU+JREp2uz8FYDpAwFsC/3bGWVrKW6aUW6T
sAoQwdcL2XlokQ3NkdwimYv6gB2sP8kNje7G8gl1kEodu6QucDwChvHHTF5MQuPz
5L2iWBif2ZFBE7+AokIYL3rZCUSVviq6e77hJ2p0l4Rtx2I20AaHKaOFGITwtELx
mIVzJ9747lEq6xV9VeFFG7wmJqFioy39hPoKL9rtMhkGg78LsX6famSjYqUW12qs
QjyazOUd0ve3C8WBX/PVIA3I3WpQTpgT/xIZSbuHUa7fYE5+BRvcZOaRVk6WfHja
```

```
ABCx43D5f6FPHj3pfcsGZzKrT26QJYeIIcNgsgH+KQAszjfIYF1iJNCkJQGGkbQX
DL+Fvr7PaLOWjf31A85KSHWXe9F7cFovwk+l6Q8RuT4al8sL9ibYcwIDAQABo3Iw
cDBPBgNVHREESDBGghtjYXNkZXYtc2FtbHNwLm5ld3NjaG9vbC5lZHWGJ2h0dHBz
Oi8vY2FzZGV2Lm5ld3NjaG9vbC5lZHUvc2hpYmJvbGV0aDAdBgNVHQ4EFgQUlo8B
+EW+irsywPjzuc8cBeLvXU8wDQYJKoZIhvcNAQELBQADggGBAKh7J3VZAYNs5Lr3
ox7rvz9Vhx5ZzvZ28j6TlTFvtbh4OSNYlP3G3o627MtORY/bPzm63bQrfq53OTOf
JiqMENEOGXrqBbFqRfR32P75BWfXsh2ZSxdGXU6O9czFpjrycAKZgWv9U2OYFpyb
m14RzSqQq34pyL8nScJ2dO5cK/Ei5SeL76U8Jf68fVVhL6eEZ3eV93jtUafLW4r6
ouiO9v26d+W6hnzk5R0ntitNgkCudcpFH/heDXawbxhXOkEHbAQ1ggqdE/vnWVfx
PKxvhKw6UW1PsaYoy+yiXwzX3rIxLGTfnqJ8cdb4gjDE8tXOxMjVdbWeFTY5aDWG
Nv9KkJ9W5dEz1s91enceM9XzINvca2d/qa7hflEEHnWNhXKIQ8BEXKjEPFFkcclo
r5hpkVNVVGEC2ZA1nSQlrDBMpf/3uDgPxVh474vuDTJHvcEXpfNqEITBdxO2t7dV
HGiljHQvACTaqHAL0sYUpZPVk3CE2RV+B0m3jugIlnT3VB7tFQ==
</ds:X509Certificate></ds:X509Data></ds:KeyInfo></md:KeyDescriptor><m
d:KeyDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" us
e="encryption"><ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsi
g#"><ds:KeyName>casdev-samlsp.newschool.edu</ds:KeyName><ds:KeyName>h
ttps://casdev.newschool.edu/shibboleth</ds:KeyName><ds:X509Data><ds:X
509SubjectName>CN=casdev-samlsp.newschool.edu</ds:X509SubjectName><d
s:X509Certificate>MIIEQTCCAqmgAwIBAgIJAJgyfve+2p4MMA0GCSqGSIb3DQEBCwU
AMCYxJDAiBgNV
BAMTG2Nhc2Rldi1zYW1sc3AubmV3c2Nob29sLmVkdTAeFw0xNzEwMjUxOTA5NTNa
Fw0yNzEwMjMxOTA5NTNaMCYxJDAiBgNVBAMTG2Nhc2Rldi1zYW1sc3AubmV3c2No
b29sLmVkdTCCAaIwDQYJKoZIhvcNAQEBBQADggGPADCCAYoCggGBAKnAVWGK7S1f
nmL41kvXlU9l2BDQNKEpoEnu428Bg8Az/5e1wxs2eMk9XymPobEJ5LlT0Fyr4nSl
NqRSHkCOFdoll+W8qfqn3AxgYaGDccU+JREp2uz8FYDpAwFsC/3bGWVrKW6aUW6T
sAoQwdcL2XlokQ3NkdwimYv6gB2sP8kNje7G8gl1kEodu6QucDwChvHHTF5MQuPz
5L2iWBif2ZFBE7+AokIYL3rZCUSViq6e77hJ2p0l4Rtx2I20AaHKaOFGITwtELx
mIVzJ9747lEq6xV9VeFFG7wmJqFioy39hPoKL9rtMhkGg78LsX6famSjYqUW12qs
QjyazOUd0ve3C8WBX/PVIA3I3WpQTpgT/xIZSbuHUa7fYE5+BRvcZOaRVk6WfHja
ABCx43D5f6FPHj3pfcsGZzKrT26QJYeIIcNgsgH+KQAszjfIYF1iJNCkJQGGkbQX
DL+Fvr7PaLOWjf31A85KSHWXe9F7cFovwk+l6Q8RuT4al8sL9ibYcwIDAQABo3Iw
cDBPBgNVHREESDBGghtjYXNkZXYtc2FtbHNwLm5ld3NjaG9vbC5lZHWGJ2h0dHBz
Oi8vY2FzZGV2Lm5ld3NjaG9vbC5lZHUvc2hpYmJvbGV0aDAdBgNVHQ4EFgQUlo8B
+EW+irsywPjzuc8cBeLvXU8wDQYJKoZIhvcNAQELBQADggGBAKh7J3VZAYNs5Lr3
ox7rvz9Vhx5ZzvZ28j6TlTFvtbh4OSNYlP3G3o627MtORY/bPzm63bQrfq53OTOf
JiqMENEOGXrqBbFqRfR32P75BWfXsh2ZSxdGXU6O9czFpjrycAKZgWv9U2OYFpyb
m14RzSqQq34pyL8nScJ2dO5cK/Ei5SeL76U8Jf68fVVhL6eEZ3eV93jtUafLW4r6
ouiO9v26d+W6hnzk5R0ntitNgkCudcpFH/heDXawbxhXOkEHbAQ1ggqdE/vnWVfx
PKxvhKw6UW1PsaYoy+yiXwzX3rIxLGTfnqJ8cdb4gjDE8tXOxMjVdbWeFTY5aDWG
Nv9KkJ9W5dEz1s91enceM9XzINvca2d/qa7hflEEHnWNhXKIQ8BEXKjEPFFkcclo
r5hpkVNVVGEC2ZA1nSQlrDBMpf/3uDgPxVh474vuDTJHvcEXpfNqEITBdxO2t7dV
HGiljHQvACTaqHAL0sYUpZPVk3CE2RV+B0m3jugIlnT3VB7tFQ==
</ds:X509Certificate></ds:X509Data></ds:KeyInfo></md:KeyDescriptor><S
tatus><OK/></Status></StatusHandler>
casdev-samlsp#
```

The details of the XML document returned by this command aren't terribly important since the SP hasn't been configured yet.

> **Note:** The status page is protected by an IP address ACL, so the `curl` command must be run from the server against the `localhost` IP address (`127.0.0.1`). The `-k` option to `curl` is also needed, since the server's SSL certificate is advertising a different host name.

References

- Shibboleth SP: NativeSPLinuxRPMInstall
- ShibInstallFest: Linux Service Provider (RHEL 7.0)

Configure HTTPD to use the SP

Now that the Shibboleth SP has been installed, the `mod_shib` Apache HTTPD module can be configured and some web content can be created to secure with it.

> ℹ **Note:** The steps in this section should be performed on the client server (***casdev-samlsp***), not the master build server (***casdev-master***).

Configure `mod_shib` settings

Edit the file `/etc/httpd/conf.d/shib.conf` (installed as part of the `yum` package) and locate the `<Location>` tag (around line 49), which should look something like this:

```
<Location /secure>
```

Change the path of the directory to be secured to `/secured-by-saml`:

```
<Location /secured-by-saml>
```

Restart HTTPD

Run the command

```
casdev-samlsp# systemctl restart httpd
```

to restart the HTTPD server with the new configuration.

Create example content

Edit the file `/var/www/html/index.php` and replace the call to `phpinfo()` with another link, like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1">
    <link rel="stylesheet"
      href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.m
in.css">
  </head>
  <body>
    <div class="container">
      <h1>Hello, World!</h1>
        <p><big>The quick brown fox jumped over the lazy dogs.</bi
g></p>
        <p><big>Click <a href="secured-by-saml/index.php">here</a> fo
r some
          content secured by username and password.</big></p>
    </div>
  </body>
</html>
```

Then create a directory, `/var/www/html/secured-by-saml`, and create the file `/var/www/html/secured-by-saml/index.php` with the following contents:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1">
    <link rel="stylesheet"
      href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.m
in.css">
  </head>
  <body>
    <div class="container">
      <h1>Secured Content</h1>
      <p><big>This is some secure content. You should not be able to
see it
       until you have entered your username and password.</big></p>
      <h2>Attributes Returned by SAML</h2>
      <?php
        echo "<pre>";

        if (array_key_exists('REMOTE_USER', $_SERVER)) {
            echo "REMOTE_USER = " . $_SERVER['REMOTE_USER'] . "<br>";
        }

        foreach ($_SERVER as $key => $value) {
            if (strpos($key, 'SAML_') === 0) {
                echo substr($key, 5) . " = " . $value . "<br>";
            }
        }

        echo "</pre>";
      ?>
    </div>
  </body>
</html>
```

The PHP code here will display environment variables that are used by `mod_shib` to pass attributes returned by the SAML IdP along to the web application.

Configure the SP

Now that the Shibboleth SP has been installed, the `shibd` daemon can be configured to communicate with the CAS SAML IdP.

> ℹ **Note:** The steps in this section should be performed on the client server (***casdev-samlsp***), not the master build server (***casdev-master***).

Configure the SAML entity and IdP settings

Edit the file `/etc/shibboleth/shibboleth2.xml` and make the changes described below to set the SP's *entityID* (the string the SP uses to identify itself to the IdP) and tell it which IdP to use.

Set the entityID

Locate the `<ApplicationDefaults>` XML tag (around line 23) and change the value of the `entityID` attribute to reflect the URL of the SAML client host (***casdev-samlsp.newschool.edu***).

```
<ApplicationDefaults entityID="https://casdev-samlsp.newschool.edu/sh
ibboleth"
                     REMOTE_USER="eppn persistent-id targeted-id">
```

Set the `REMOTE_USER` attribute and attribute prefix

On the next line, change the value of the `REMOTE_USER` attribute to `uid`, and add a new attribute, `attributePrefix`, as shown:

```
<ApplicationDefaults entityID="https://casdev-samlsp.newschool.edu/sh
ibboleth"
                     REMOTE_USER="uid" attributePrefix="SAML_">
```

The `REMOTE_USER` attribute specifies which user attribute, returned by the IdP, should be used to populate the `REMOTE_USER` environment variable for the web application to access. The `attributePrefix` attribute specifies a prefix string to be applied to all the environment variables set by the `mod_shib` plugin, including the environment variables containing user attribute values.

Configure session security

Locate the `<Sessions>` XML tag (around line 35) and make sure the value of the `handlerSSL` attribute is set to `true`, and the value of the `cookieProps` attribute is set to `https`:

```
<Sessions lifetime="28800" timeout="3600" relayState="ss:mem"
          checkAddress="false" handlerSSL="true" cookieProps="https">
```

These settings will ensure that all sessions between the SP and the IdP are encrypted with TLS/SSL, and that cookies cannot be exchanged over insecure channels.

Point the SP to the IdP

Locate the `<SSO>` XML tage (around line 44) and change the value of the `entityID` attribute to the URL of the CAS SAML IdP. Delete the `discoveryProtocol` and `discoveryURL` attributes; they are not needed for this configuration.

```
<SSO entityID="https://casdev.newschool.edu/cas/idp">
  SAML2 SAML1
</SSO>
```

Tell the SP where to get the IdP's metadata

Locate the (commented out) examples of `MetadataProvider` definitions (around lines 73-90), and insert the following below them:

```
<MetadataProvider type="XML" validate="true"
      uri="https://casdev.newschool.edu/cas/idp/metadata"
      backingFilePath="casdev-metadata.xml" reloadInterval="7200">
</MetadataProvider>
```

This tells the SP what URL to use to obtain the IdP's metadata, gives it a file name in which to store it, and a time limit after which it should be reloaded from the server. (The metadata backing file will be stored in `/var/cache/shibboleth`.)

Configure attribute processing

A SAML IdP sends user attributes to a SAML SP in the form of SAML assertions. To avoid misinterpretation, every attribute has a unique identifier, agreed upon by standards-setting bodies. This identifier (name) is different from the attribute names used by back-end data stores and consuming applications. For example, a telephone number might be returned from the IdP as follows:

```
<saml:Attribute FriendlyName="telephoneNumber" Name="urn:oid:2.5.4.2
0"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="xs:string">555-5555</saml:AttributeValue>
</saml:Attribute>
```

Although the back-end data store from which the IdP obtained the attribute (e.g., an LDAP directory) might refer to this attribute as a `telephoneNumber`, a consuming application might call it `telephoneNumber` or `telephone` or `phone` or something else. Therefore, to make sure that IdPs and SPs know that they're talking about the same thing, the [SAML V2.0 LDAP/X.500 Attribute Profile][https://wiki.oasis-open.org/security/SstcSaml2AttributeX500Profile] specifies that this attribute should be identified as `urn:oid:2.5.4.20` (similar values are defined for other common LDAP attributes).

To map between the standard attribute names used by the IdP and SP and the "friendly" attribute names used by applications, the Shibboleth SP uses a file called `/etc/shibboleth/attribute-map.xml`, which contains definitions like this:

```
<Attribute name="urn:oid:2.5.4.20" id="telephoneNumber"/>
<Attribute name="urn:mace:dir:attribute-def:telephoneNumber" id="tele
phoneNumber"/>
```

The `urn:mace` attribute namespace is another namespace, registered with the IETF and IANA, for Internet2's Middleware Architecture Committee for Education. It is heavily used by Internet2 and InCommon member organizations.

Enable LDAP attribute mappings

To enable the pre-defined LDAP attribute mappings in the SP, edit the file `/etc/shibboleth/attribute-map.xml` and remove the comment lines (`<--` and `-->`) around the section labeled “Examples of LDAP-based attributes” (around lines 92-149).

Add custom attribute mappings

When we configured attribute resolution (page 144) in the CAS server, we configured a number of standard LDAP attributes, but also a couple of non-standard ones, `role` and `UDC_IDENTIFIER`. To tell the Shibboleth SP how to process these attributes, edit the file `/etc/shibboleth/attrbute-map.xml` and add the following lines to the end (before the `</Attributes>` XML close tag):

```
<Attribute name="urn:newschool:attribute-def:role" id="role"/>
<Attribute name="urn:newschool:attribute-def:UDC_IDENTIFIER" id="UD
C_IDENTIFIER"/>
```

We cannot use the `urn:oid` or `urn:mace` namespaces, since those are controlled by standards bodies. So instead, we define our own namespace, `urn:newschool`, modeled after `urn:mace`.

Restart `shibd`

Run the command

```
casdev-samlsp# systemctl restart shibd
```

to restart the Shibboleth daemon with the new configuration.

References

- Shibboleth SP: NativeSPConfiguration
- ShibInstallFest: Linux Service Provider (RHEL 7.0)
- Shibboleth: AttributeNaming

Update the service registry

Just like CAS-enabled services, SAML-enabled services must be defined in the service registry.

Create a service definition for the SAML client

Create the file `etc/cas/services/ApacheSecuredBySAML-1509030300.json` (replace `1509030300` with the current `date +%s` or `YYYYMMDDhhmmss` value) in the `cas-overlay-template` directory on the master build server (***casdev-master***) with the following contents:

```
{
  "@class" : "org.apereo.cas.support.saml.services.SamlRegisteredService",
  "serviceId" : "https://casdev-samlsp.newschool.edu/shibboleth",
  "name" : "Apache Secured By SAML",
  "id" : 1509030300,
  "description" : "CAS development Apache mod_shib/shibd server with username/password protection",
  "metadataLocation" : "https://casdev-samlsp.newschool.edu/Shibboleth.sso/Metadata",
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnMappedAttributeReleasePolicy",
    "allowedAttributes" : {
      "@class" : "java.util.TreeMap",
      "cn" : "urn:oid:2.5.4.3",
      "displayName" : "urn:oid:2.16.840.1.113730.3.1.241",
      "givenName" : "urn:oid:2.5.4.42",
      "mail" : "urn:oid:0.9.2342.19200300.100.1.3",
      "role" : "urn:newschool:attribute-def:role",
      "sn" : "urn:oid:2.5.4.4",
      "uid" : "urn:oid:0.9.2342.19200300.100.1.1",
      "UDC_IDENTIFIER": "urn:newschool:attribute-def:UDC_IDENTIFIER"
    }
  },
  "evaluationOrder" : 1125
}
```

This is similar to the service definitions created previously, but there are some differences:

1. The `@class` of the service is

`org.apereo.cas.support.saml.services.SamlRegisteredService` rather than `org.apereo.cas.services.RegexRegisteredService`.

2. The `serviceId` is specified as an exact-match string, not a regular expression. Specifically, this attribute must be equal to the entityID of the service.
3. The new attribute, `metadataLocation`, is used to tell the IdP where it can obtain the SP's metadata. This will be automatically retrieved and stored in `/etc/cas/saml/metadata-backups/` when the SP first connects to the IdP. For SPs that do not provide a URL from which to obtain metadata, the metadata can be obtained by other means (e.g., email from the service provider) and saved to a file which can then be identified in this attribute with a `file:` URI.
4. The `ReturnMappedAttributeReleasePolicy` is used to assign the SAML-specific attribute names expected by the SP to the attributes. The values to be used here can be obtained from the SP's `/etc/shibboleth/attribute-map.xml` file.

References

- CAS 5: JSON Service Registry
- CAS 5: SAML2 Authentication

Install and test the application

Before the SAML client application can be used, the the updated CAS server configuration files must be installed. Then everything can be tested by accessing the “public” part of the client application web site and then clicking on the link to the “secure” section. At that point, the browser should be redirected to the CAS server, where upon successful authentication the secure content, including the values of the attributes, will be displayed.

Because both the load balancer and the CAS server use cookies, it’s usually best to perform testing with an “incognito” or “private browsing” instance of the web browser that deletes all cookies each time it is closed.

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Shut down all but one of the pool servers

Operating CAS with a pool of servers instead of a single server requires special configuration. Because that configuration hasn't been completed yet, testing must be performed against a single server. Therefore, the other servers in the pool should be shut down so that the load balancer will direct all traffic to that single server. Run the command

```
# systemctl stop tomcat
```

on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

Access the public site

Open up a web browser (in "incognito" or "private browsing" mode) and enter the URL of the SAML-enabled web site:

```
https://casdev-samlsp.newschool.edu/
```

The contents of `/var/www/html/index.php` should be displayed, looking something like this:

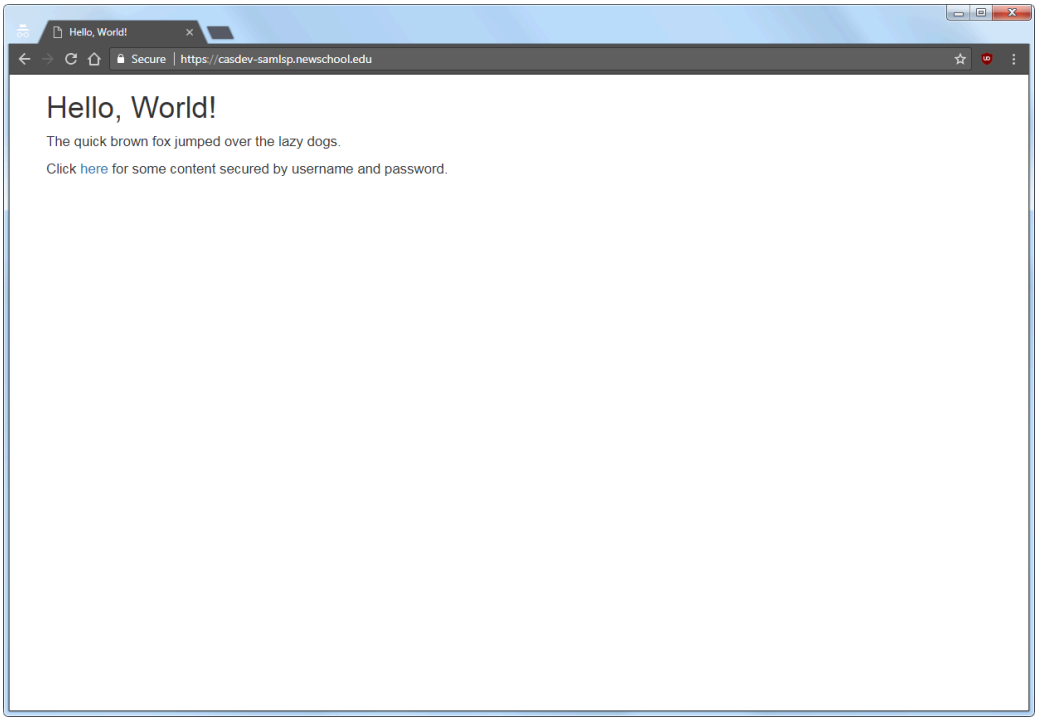

Figure 18. The "public" site

Access the secure area

Click on the “here” link to access the secure content, and you will be redirected to the CAS server login page, as shown below:

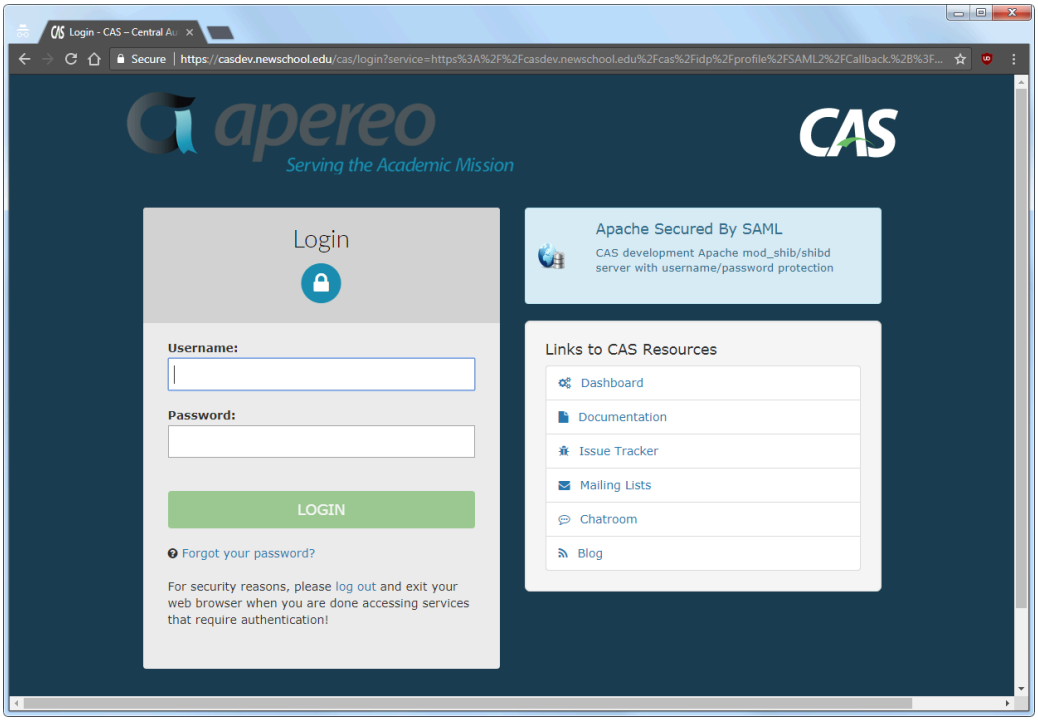

Figure 19. The CAS login page

Note that the contents of the `name` field from the service registry are displayed at the top of the right-hand column; make sure that `Apache Secured by SAML` is displayed here. Enter a valid username and password (Active Directory or LDAP) that is also registered with Duo and, upon successful authentication, the contents of `/var/www/html/secured-by-saml/index.php` will be displayed:

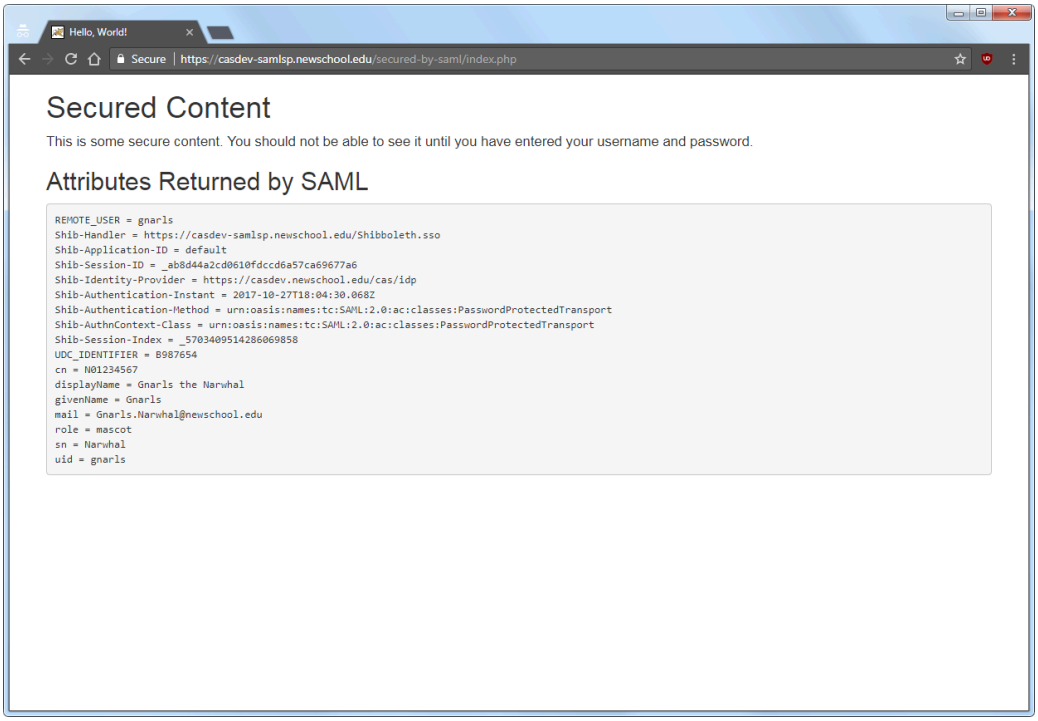

Figure 20. The "secure" content

Restart the pool servers

One testing is complete, run the command

```
# systemctl start tomcat
```

on each of the pool servers shut down previously.

Adding MFA to SAML authentication

Adding a multi-factor authentication flow to a SAML-authenticated service is as easy as editing the service registry definition to add the `multifactorPolicy` directive. For example, the service registry definition file created in this chapter would look like this with Duo multi-factor authentication added:

```
{
  "@class" : "org.apereo.cas.support.saml.services.SamlRegisteredServ
ice",
  "serviceId" : "https://casdev-samlsp.newschool.edu/shibboleth",
  "name" : "Apache Secured By SAML",
  "id" : 20171026110500,
  "description" : "CAS development Apache mod_shib/shibd server with
username/password protection",
  "metadataLocation" : "https://casdev-samlsp.newschool.edu/Shibbolet
h.sso/Metadata",
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnMappedAttributeReleaseP
olicy",
    "allowedAttributes" : {
      "@class" : "java.util.TreeMap",
      "cn" : "urn:oid:2.5.4.3",
      "displayName" : "urn:oid:2.16.840.1.113730.3.1.241",
      "givenName" : "urn:oid:2.5.4.42",
      "mail" : "urn:oid:0.9.2342.19200300.100.1.3",
      "role" : "urn:newschool:attribute-def:role",
      "sn" : "urn:oid:2.5.4.4",
      "uid" : "urn:oid:0.9.2342.19200300.100.1.1",
      "UDC_IDENTIFIER": "urn:newschool:attribute-def:UDC_IDENTIFIER"
    }
  },
  "multifactorPolicy" : {
    "@class" : "org.apereo.cas.services.DefaultRegisteredServiceMulti
factorPolicy",
    "multifactorAuthenticationProviders" : [ "java.util.LinkedHashSe
t", [ "mfa-duo" ] ]
  },
  "evaluationOrder" : 1125
}
```

Testing SAML and MFA together

In the service registry, CAS-enabled services are identified by URL. Thus, we were able to create CAS-only and CAS-plus-MFA services on the CAS client server simply by creating different directories in `/var/www/html`, resulting in two different URLs and two different service registry definitions, one with MFA enabled and one without.

SAML services are a little different, though. They're identified by an entityID, and generally there's only one entityID per application (recall that this value was set in the SP's configuration file, `/etc/shibboleth/shibboleth2.xml`). Thus, simply creating a second directory in `/var/www/html` won't work in and of itself, because the SP software will present the same entityID to the CAS SAML IdP for both directories. It *is* possible to configure the Shibboleth SP to present different entityID values under different conditions (such as different directories), but doing so is a complicated, multi-step undertaking that frankly isn't worth the effort.

Instead, to test SAML and MFA together, just add the `multifactorPolicy` attribute to the existing service definition and test. To disable MFA, just comment that part of the definition out:

```
/*
"multifactorPolicy" : {
  "@class" : "org.apereo.cas.services.DefaultRegisteredServiceMultifa
ctorPolicy",
  "multifactorAuthenticationProviders" : [ "java.util.LinkedHashSe
t", [ "mfa-duo" ] ]
},
*/
```

References

- CAS 5: Duo Security Authentication
- CAS 5: Configuration Properties: Multi-factor Authentication

Commit changes to Git

Before moving on to building the SAML client, commit the new service registry definition file to Git to make changes easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# git add etc/cas/services/ApacheSecuredBySAML-1509030300.json
casdev-master# git commit -m "Set up SAML client"
[newschool-casdev 6ad660c] Set up SAML client
 1 file changed, 29 insertions(+)
 create mode 100644 etc/cas/services/ApacheSecuredBySAML-1509030300.json
casdev-master#
```

on the master build server (***casdev-master***).

Enabling the dashboard (admin pages)

Summary: The dashboard will be enabled to support server administration and monitoring.

CAS 5 provides about two dozen endpoints under the `${cas.server.prefix}/status/` URL that allow administrators, permissions granting, to obtain real-time configuration data and performance monitoring statistics, and also make configuration changes. It also provides a rudimentary dashboard interface through which these endpoints can be accessed, as shown in Figure 21.

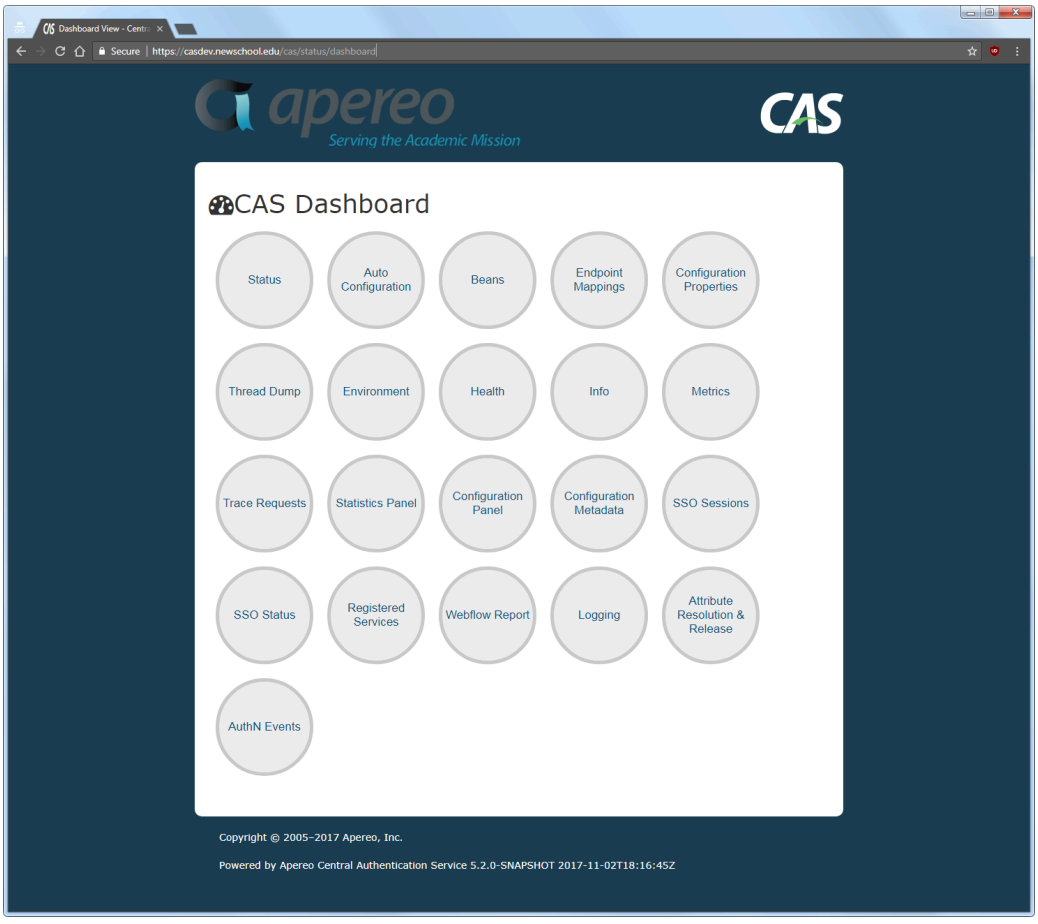

Figure 21. The dashboard

Some of the more “interesting” endpoints include:

- **Status** – A quick text-only summary of the server’s health, number of sessions, memory usage, host name, server name, and CAS version.
- **Configuration Properties** – A JSON-formatted dump of all the CAS

configuration properties and their current values.

- **Statistics Panel** – A dashbnoard-like display of expired and unexpired tickets, JVM statistics, and other information.
- **SSO Sessions** – A dashboard-like display of current active sessions, including usernames, tickets, authentication times, etc.
- **Registered Services** – A JSON-formatted dump of the service registry (the management webapp (page 236) will provide a better representation of this information).
- **Attribute Resolution and Release** – An interactive interface to test attribute resolution for individual users, and attribute release to specific services.

As of this writing, most of the endpoints don't do anything beyond print information in raw JSON format. However, even this can be useful for examining the current server state, although it may require copying-and-pasting the output into a JSON pretty-printer (such as Code Beautify) to make any sense of it.

References

- CAS 5: Monitoring and Statistics

Configure admin pages properties

There are two types of endpoints supported by the CAS server, those that can be viewed and managed via the dashboard only, and those that can also be viewed and managed with the Spring Boot Administration server. Since we will not be using the Spring Boot Administration server in our deployment, the distinction is somewhat unimportant, except that it introduces a few extra configuration settings.

Enable all the endpoints

Edit the file `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and add the following lines to enable the endpoints:

```
cas.adminPagesSecurity.actuatorEndpointsEnabled:        true
cas.monitor.endpoints.enabled:          true
endpoints.enabled:                      true
```

This will enable both the CAS-style endpoints and the Spring Boot-style endpoints. The above settings apply globally to all the endpoints; it is also possible to enable and disable each endpoint individually; see the configuration property references below.

Configure endpoint security

The top-level `/status` endpoint is always secured by an IP address pattern (regular expression). If no other security is configured, all the endpoints underneath `/status` are also secured by that pattern. The endpoints underneath `/status` may also be secured by the CAS server itself, just like any other CAS-enabled service, or they may secured with Spring Security, which supports basic authentication (master username and password), database-based authentication, LDAP-based authentication, and other methods. If neither CAS nor Spring Security are configured to secure the endpoints underneath `/status`, then their security will rely solely on the IP addrees pattern as well.

For our installation, we will use the IP address pattern to secure the `/status` endpoint, and the CAS server to secure the endpoints underneath it.

Edit the file `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and make the changes in the following sections.

Configure the IP address pattern

Locate the line for the `cas.adminPagesSecurity.ip` property which, as distributed with the CAS Maven WAR overlay, allows access only from the local host:

```
cas.adminPagesSecurity.ip=127\.0\.0\.1
```

The value of this property is a regular expression that is matched against the IP address of the incoming request. If there is a match, access is granted; if there is no match, access is denied. Change the setting to allow access from whatever address(es) should have access. In our case, we will allow access from (a) the IT department office subnet (`192.168.50.0/24`), and (b) the internal interfaces of the F5 load balancers (`192.168.1.10` , `192.168.1.20`) (more on the reason for this later):

```
cas.adminPagesSecurity.ip:              ^192\\.168\\.(5
0\\.[0-9]{1,3}|1\\.[12]0)$
```

The backslashes are doubled because the backslash is a special character in Java properties files; doubling them results in a backslash actually being put into the regular expression (which causes the '.' to match a literal '.' instead of any character).

Mark the endpoints "not sensitive"

In addition to enabling/disabling endpoints, CAS allows each endpoint to be marked "sensitive" or "not sensitive." The term "sensitive" is a little confusing in this context (it comes from the Spring Security world). Basically, if an endpoint is marked "sensitive" then Spring Security will be used to control access to it. If the endpoint is not marked "sensitive" then CAS will be used to secure it. Since we want to use the CAS server to secure all the endpoints, add the following properties:

```
cas.monitor.endpoints.sensitive:        false
endpoints.sensitive:                    false
```

Configure the CAS server settings

To use the CAS server to secure the endpoints, the URL of the CAS server's login page and the name of the service to be accessed must be configured. In addition, to limit access to the endpoints to a specific set of "administrator" users, a separate user file should be provided that lists these users and their roles. To do this, add the following properties:

```
cas.adminPagesSecurity.loginUrl:        ${cas.server.prefix}/login
cas.adminPagesSecurity.service:         ${cas.server.prefix}/status/d
ashboard
cas.adminPagesSecurity.users:           file:/etc/cas/config/admuser
s.properties

cas.adminPagesSecurity.adminRoles[0]:   ROLE_ADMIN
```

Create the `admusers.properties` file

Create a file called `etc/cas/config/admusers.properties` in the `cas-war-overlay` directory on the master build server (***casdev-master***) that looks like this:

```
# This file lists the users who are allowed access to the CAS /statu
s/*
# endpoints ("adminpages").
#
# The syntax for each line is:
#
# username=password,grantedAuthority[,grantedAuthority][,enabled|disa
bled]
#
gnarls=passwordnotused,ROLE_ADMIN
```

Additional users can be listed, one per line, below the first one. Since the users are authenticating to the CAS server (against Active Directory or LDAP) the password field is not needed, and can be filled with any string (e.g., `passwordnotused`). Each user may have up to two `grantedAuthority` values assigned; for the user to be able to access the endpoints, one of these values must match one of the values

assigned to the `cas.adminPagesSecurity.adminRoles` property (see above). This file is also used by the management webapp (page 236), so having two possible values allows users to be given access to the endpoints, the webapp, or both.

References

- CAS 5: Configuration Properties: CAS Endpoints
- CAS 5: Configuration Properties: Spring Boot Endpoints

Update the service registry

Since we are using the CAS server to protect the endpoints, we need to create a service definition for the dashboard.

Create a service definition for the dashboard

Create the file `etc/cas/services/CASAdminDashboard-1509646291.json` (replace `1509646291` with the current `date +%s` or `YYYYMMDDhhmmss` value) in the `cas-overlay-template` directory on the master build server (***casdev-master***) with the following contents:

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^https://casdev.newschool.edu/cas/status/dashboar
d(\\z|/.*)",
  "name" : "CAS Admin Dashboard",
  "id" : 1509646291,
  "description" : "CAS dashboard and administrative endpoints",
  "evaluationOrder" : 5000
}
```

There is no need to create service definitions for all the other endpoints underneath `/status`; they will all be authenticated by this one. Note, however, that any attempt to access those other endpoints before accessing the `/status/dashboard` endpoint and authenticating to the CAS server will fail.

References

- CAS 5: JSON Service Registry

Install and test the application

Before the dashboard can be used, the the updated CAS server configuration files must be installed.

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Shut down all but one of the pool servers

Operating CAS with a pool of servers instead of a single server requires special configuration. Because that configuration hasn't been completed yet, testing must be performed against a single server. Therefore, the other servers in the pool should be shut down so that the load balancer will direct all traffic to that single server. Run the command

```
# systemctl stop tomcat
```

on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

Access the dashboard

Open up a web browser (in "incognito" or "private browsing" mode) and enter the URL of the dashboard:

```
https://casdev.newschool.edu/cas/status/dashboard
```

The dashboard should be displayed, as shown previously in Figure 21:

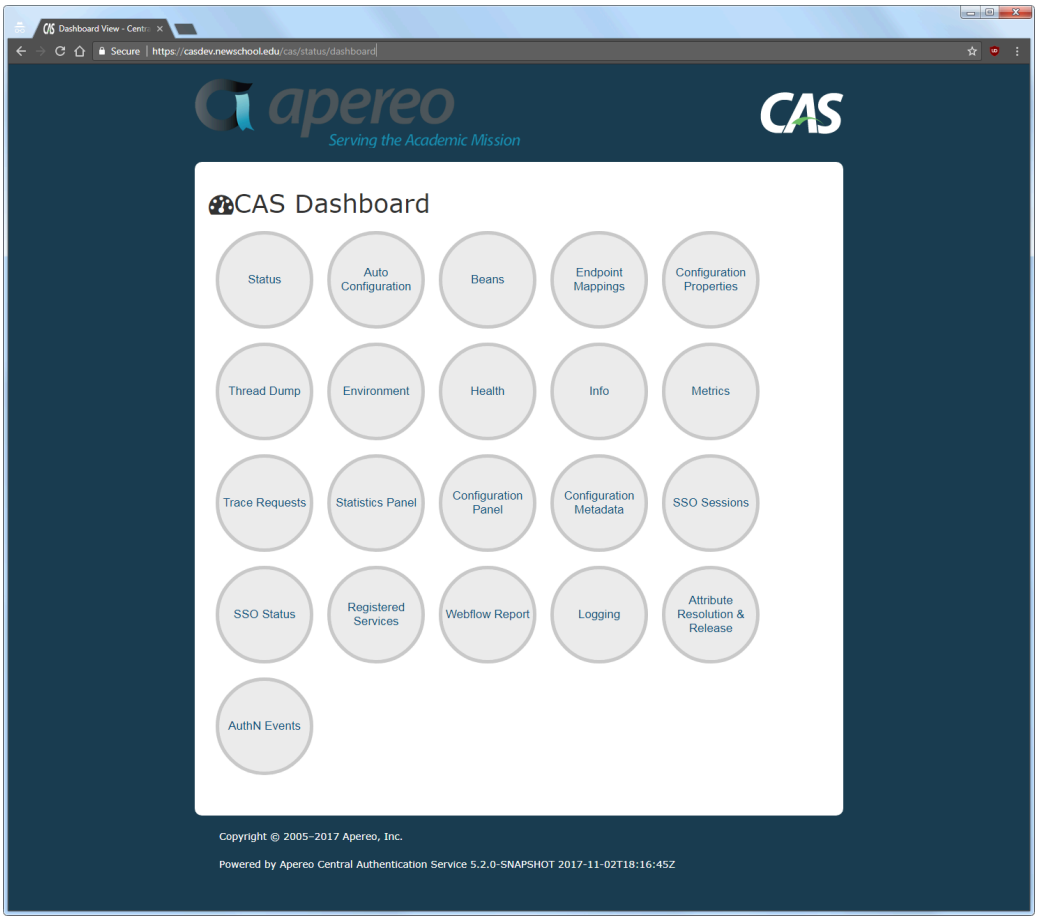

Figure 21. The dashboard

Click on the various circles on the dashboard to see their contents.

Restart the pool servers

One testing is complete, run the command

```
# systemctl start tomcat
```

on each of the pool servers shut down previously.

Update the load balancer service monitor

When we first configured the load balancers (page 66), we defined a monitor for the server pool that connected to each server every 5 seconds and issued a `GET /` HTTP request to verify that Tomcat was running. Later, when we built and installed the CAS server (page 99), we adjusted this monitor to instead issue a `GET /cas/login` request and check for some text on the login page, to verify that not only Tomcat, but the CAS application itself, were running. The drawback to this change is that it resulted in a huge number of started-but-never-finished login sessions on the CAS servers, and a correspondingly huge amount of output in the log files (which, in addition to wasting disk space, made the logs much harder to read).

However, now that we have the `/status` endpoint configured, we can adjust the monitor to issue a `GET /cas/status` request instead. The output from a `GET /cas/status` request looks something like this:

```
Health: OK

        1.SessionMonitor: OK - 2 sessions. 0 service tickets.

        2.MemoryMonitor: OK - 1648.57MB free (90.55%), 172.04MB use
d, 1820.61MB total.

Host:                casdev-srv01
Server:                https://casdev.newschool.edu
Version:        5.2.0-SNAPSHOT
```

So, by checking the data returned from the server for `Health: OK`, we can still verify that Tomcat and the CAS server are running. To do this, log into the F5 administration portal and edit the `casdev_https_8443_monitor`. Change these two lines in the definition:

```
recv "Login - CAS"
send "GET /cas/login\\r\\n"
```

to:

```
recv "Health: OK"
send "GET /cas/status\\r\\n"
```

so that the entire monitor looks like this:

```
ltm monitor https /Common/casdev_https_8443_monitor {
    adaptive disabled
    cipherlist DEFAULT:+SHA:+3DES:+kEDH
    compatibility enabled
    defaults-from /Common/https
    description "Cas Dev Application HTTPS Monitor"
    destination *:8443
    interval 5
    is-dscp 0
    recv "Health: OK"
    recv-disable none
    send "GET /cas/status\\r\\n"
    time-until-up 0
    timeout 16
}
```

With the new monitor, there will still be a single line written to the Tomcat access log, but there will not be any more lines written to the CAS log file.

Commit changes to Git

Before moving on to the next task, commit the changes made to `cas.properties`, as well as the new user file and new service registry definition file to Git to make changes easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git add etc/cas/config/admusers.properties
casdev-master# git add etc/cas/services/CASAdminDaashboard-150964629
1.json
casdev-master# git commit -m "Enabled the admin status dashboard"
[newschool-casdev 4e11f58] Enabled the admin status dashboard
 3 files changed, 52 insertions(+), 2 deletions(-)
 create mode 100644 etc/cas/config/admusers.properties
 create mode 100644 etc/cas/services/CASAdminDashboard-1509646291.jso
n
casdev-master#
```

on the master build server (***casdev-master***).

Building the management webapp

Summary: Build and install the separate management webapp to make it easier to manage the service registry and prepare for managing the service registry in a high availability environment.

The CAS management webapp is a web-based GUI that allows CAS administrators to create, modify, and delete service definitions in the service registry. It is implemented as a completely separate web application built independently of the CAS server. It can be deployed in the same Java Servlet container that is running the CAS server, or it can be deployed in a completely separate container. Either way, the operation of the CAS server does not depend in any way on the state of the management webapp.

The management webapp isn't strictly needed when using a file-based service registry such as the JSON registry we have been using to this point, although it will help avoid JSON syntax errors and similar mistakes. Where the webapp becomes important is when using storage back ends such as databases for the service registry. In that configuration, the webapp becomes the interface fronting the CRUD operations that deal with the back end storage system.

Since implementing high availability (page 321) will require replacing the JSON file-based service registry with something else that is suited for use in a multiple server, clustered environment, the webapp is going to be a key component of our environment.

References

- CAS 5: Services Management Webapp

Create a Maven WAR overlay project

The CAS project provides a separate Maven WAR overlay template project for building the management webapp. We will use that as the starting point for our project.

Clone the overlay template project

Use Git to clone the overlay template project from GitHub. Run the commands

```
casdev-master# cd /opt/workspace
casdev-master# git clone https://github.com/apereo/cas-management-overlay.git
Cloning into 'cas-management-overlay'...
remote: Counting objects: 297, done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 297 (delta 8), reused 14 (delta 4), pack-reused 276
Receiving objects: 100% (297/297), 132.54 KiB | 0 bytes/s, done.
Resolving deltas: 100% (152/152), done.
casdev-master#
```

on the master build server (***casdev-master***). This will make a local copy of all files in the template project and store them in a directory called `cas-management-overlay`. It will also record the information needed for Git to keep the local copy of the files synchronized with the copy stored on GitHub, so that corrections and updates made by the project team can be incorporated into our project from time to time.

> ☑ **Tip:** As an alternative to using Git to clone a repository, GitHub allows the files in a repository to be downloaded in a Zip archive. However, this method does not include the metadata that Git needs to keep the local copy in sync with the master repository.

Switch to the right branch

The GitHub repository for the overlay template project contains multiple versions of the template; each version is stored as a separate branch of the project. The `master` branch usually points to the version of the template used for configuring and deploying the latest stable release of the CAS server; this is the branch that will initially be copied to disk by cloning the project. Run the commands

```
casdev-master# cd cas-management-overlay
casdev-master# grep '<cas.version>' pom.xml
        <cas.version>5.2.0</cas.version>
casdev-master#
```

to determine which version of the CAS server the `master` branch will build. In most circumstances (including this project), the `master` branch of the template is the one you want to use (skip ahead to the next section, Create a local branch (page 239)).

If the `master` version of the template isn't for the version of the CAS server you want to work with (for example, if you want to work with an older version, or experiment with the version currently under development), run the command

```
casdev-master# git branch -a
* master
  remotes/origin/4.1
  remotes/origin/4.2
  remotes/origin/5.0
  remotes/origin/5.1
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
casdev-master#
```

to obtain a list of available branches, and then run the `git checkout` command to switch to that branch. For example, to switch back to the `5.1` branch, run the command

```
casdev-master# git checkout 5.1
Branch 5.1 set up to track remote branch 5.1 from origin.
Switched to a new branch '5.1'
casdev-master#  grep '<cas.version>' pom.xml
        <cas.version>5.1.5</cas.version>
casdev-master#
```

to switch branches (it's not necessary to type the `remotes/origin/` part of the branch name). This will download additional/changed files from GitHub to the local disk. You can switch back to the current version of the template by checking out the `master` branch again:

```
casdev-master# git checkout master
Switched to branch 'master'
casdev-master#  grep '<cas.version>' pom.xml
        <cas.version>5.2.0</cas.version>
casdev-master#
```

Create a local branch

After you're on the right branch (for our project, you should be on the `master` branch), create a new branch local to your project, which will be used to track all of your changes and keep them separate from any changes made to the template by the CAS developers. This will make it easier in the future to merge upstream changes from the CAS project team into your local template without having to redo all your changes.

Choose a meaningful name for your branch, but not something likely to be duplicated by the CAS developers—for example, `newschool-casdev`. Run the commands

```
casdev-master# git checkout -b newschool-casdev
Switched to a new branch 'newschool-casdev'
casdev-master#
```

to create this new branch (replace `newschool-casdev` with the name of your branch).

Build the webapp

Before building the management webapp, the build must be configured to include the same service registry persistence method as the CAS server. This is accomplished by adding a dependency to the Maven project object model. For this project, that means adding the JSON service registry.

Add the JSON service registry to the project object model

Just as we did for the CAS server (page 108), edit the file `pom.xml` in the `cas-management-overlay` directory on the master build server (***casdev-master***) and locate the `dependencies` section (around line 69), which should look something like this:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-management-webapp</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
    </dependency>
</dependencies>
```

Insert a new dependency for the JSON service registry:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-management-webapp</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

Build the webapp

Run Maven to build the webapp according to the project object model:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-management-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output...check for errors)
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 25.841 s
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 29M/72M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

References

- CAS 5: Services Management Webapp
- CAS 5: JSON Service Registry

Configure webapp properties

Like the CAS server, the management webapp expects to find its configuration files in the operating system directory `/etc/cas`. The webapp configuration is controlled via settings in a separate properties file, `management.properties`, located in the `/etc/cas/config` directory. The Maven WAR overlay template provides a "source" for this file (which makes it easy to manage with Git).

Edit the file `etc/cas/config/management.properties` in the `cas-management-overlay` directory on the master build server (***casdev-master***) and make the changes in the following sections.

Configure CAS server information

Locate the lines for the `cas.server.name` and `cas.server.prefix` properties (around lines 3-4) and *set them to the same values* as the identical properties in the `cas.properties` file:

```
cas.server.name:                        https://casdev.newschool.edu
cas.server.prefix:                      ${cas.server.name}/cas
```

These two properties tell the webapp to use the named CAS server for authentication.

Configure webapp server name

Locate the line for the `cas.mgmt.serverName` property (around line 10) and set it to the URL of the server where the management webapp will be running. We're going to run the webapp on the CAS servers, in the same servlet container, so this property should have the same value as `cas.server.name`, above:

```
cas.mgmt.serverName:                    ${cas.server.name}
```

Configure users and roles

Like the dashboard, the management webapp uses a separate users file to list the users who should be able to access it (after authenticating through the CAS server) and the role(s) they should have. Locate the line for the `cas.mgmt.userPropertiesFile` property (around line 7) and set it the same value as the `cas.adminPagesSecurity.users` property in the `cas.properties` file:

```
cas.mgmt.userPropertiesFile:           file:/etc/cas/config/admuser
s.properties
```

As discussed previously (page 227), users in this file can have one or two roles assigned, making it possible to use different roles for the dashboard and the webapp. For simplicity, we will stick with the `ROLE_ADMIN` role, which is used out-of-the-box by both the dashboard and the webapp.

Delete the `users.properties` file

Since we will be re-using the same users file that the dashboard is using, the users file included with the Maven WAR overlay is not needed. To avoid confusion, remove it from the project with the following `git` command:

```
casdev-master# git rm etc/cas/config/users.properties
```

Delete embedded servlet container properties

The `management.properties` file included with the Maven WAR overlay includes some properties that are only applicable when running the management webapp as a Spring Boot application in an embedded servlet container. Since we are using an external servlet container, these settings are not needed, and should be deleted. Locate the lines below (around lines 12-13):

```
server.context-path=/cas-management
server.port=8443
```

Delete (or comment out) these lines.

Configure the JSON service registry

In order for the management webapp to manage the services registry used by the CAS server, it has to know where it is. Add a new property, `cas.serviceRegistry.json.location`, and set it to the same value as the property of the same name in `cas.properties`:

```
cas.serviceRegistry.json.location:     file:/etc/cas/services
```

Configure the stub attribute repository

The management webapp allows attribute release policies to be configured on a per-service basis. In order for it to know which attributes are available for release, a special “stub” attribute repository should be configured, with the names of all the attributes available for release (generally, this should be the combined set of names from all the attribute repositories configured in `cas.properties`):

```
cas.authn.attributeRepository.stub.attributes.UDC_IDENTIFIER:   UDC_I
DENTIFIER
cas.authn.attributeRepository.stub.attributes.cn:               cn
cas.authn.attributeRepository.stub.attributes.displayName:      displ
ayName
cas.authn.attributeRepository.stub.attributes.givenName:        given
Name
cas.authn.attributeRepository.stub.attributes.mail:             mail
cas.authn.attributeRepository.stub.attributes.sn:               sn
cas.authn.attributeRepository.stub.attributes.uid:              uid
```

Without this configuration, the management webapp will only be able to configure the “all” or “none” release policies; it will not be able to support picking and choosing the attributes to be released.

References

- CAS 5: Configuration Properties: Management Webapp

Configure logging settings

The management webapp includes its own Log4j configuration. As we did with the CAS server (page 97), we will move the location of the log file from the root of the web application directory to `/var/log/cas`.

Edit the file `etc/cas/config/log4j2-management.xml` in the `cas-management-overlay` directory on the master build server (***casdev-master***) and find the line that defines the `cas.log.dir` property (around line 9) and change its value to `/var/log/cas`, like this:

```
<Property name="cas.log.dir" >/var/log/cas</Property>
```

Adjust the log file rotation strategy (optional)

By default, the webapp log file will be rotated whenever its size reaches 512KB. To switch to same time-based rotation strategy we established for the CAS server, edit the `etc/cas/config/log4j2-management.xml` file again, and make the following changes:

1. In the `RollingFile` configuration for `cas.log` (around line 17), change the variable part of the `filePattern` attribute from `%d{yyyy-MM-dd-HH}-%i.log` to `%d{yyyy-MM-dd}.log` (remove the hour and sequence number from the pattern).
2. Remove (or comment out) the `OnStartupTriggeringPolicy` element (around line 21).
3. Remove (or comment out) the `SizeBasedTriggeringPolicy` element (around line 22).
4. Add the attributes `interval="1" modulate="true"` to the `TimeBasedTriggeringPolicy` element (around line 23).

The end result should look like this:

```
<RollingFile name="cas-management" fileName="${sys:cas.log.dir}/cas-management.log" append="true"
             filePattern="${sys:cas.log.dir}/cas-management-%d{yyyy-MM-dd}.log">
    <PatternLayout pattern="%d %p [%c] - %m%n"/>
    <Policies>
        <TimeBasedTriggeringPolicy interval="1" modulate="true"/>
    </Policies>
</RollingFile>
```

Warning: The configuration above assumes that there will be one, and only one, log file for each day. If a file with today's name already exists when Tomcat decides to rotate, the existing file will be ***overwritten***.

If you decide to keep the `OnStartupTriggeringPolicy` (which rotates the file whenever Tomcat starts) or the `SizeBasedTriggeringPolicy` (which rotates the file when it reaches a specified size (10MB by default)), or add some other policy, you should make sure the `filePattern` you use generates unique names if called more than once a day (e.g., by keeping the `%i` sequence number) or you will lose log data.

References

- CAS 5: Logging

Update the service registry

Since we are using the CAS server to protect the management webapp, we need to create a service definition for it.

Create a service definition for the webapp

Create the file `etc/cas/services/CASServiceManagement-1510002272.json` (replace `1510002272` with the current `date +%s` or `YYYYMMDDhhmmss` value) in the `cas-overlay-template` directory on the master build server (***casdev-master***) with the following contents:

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^https://casdev.newschool.edu/cas-managemen
t(\\z|/.*)",
  "name" : "CAS Services Management",
  "id" : 1510002272,
  "description" : "CAS services management webapp",
  "evaluationOrder" : 5500
}
```

References

- CAS 5: JSON Service Registry

Install and test the webapp

To deploy the management webapp, we have to copy the application we just built with Maven into Tomcat's `webapps` directory., And, to make sure everything stays in sync, it probably makes sense to (re)copy the CAS server application into `webapps` as well. We also have to copy the new and updated files in `etc/cas` from both the `cas-management-overlay` and `cas-overlay-template` directories.

Update the distribution `tar` file creation script

Earlier (page 99), we created a shell script to handle building the distribution `tar` file (if you didn't do this then, now is the time to do it). That script can be extended to combine the necessary components of both the `cas-overlay-template` and the `cas-management-overlay` into a single `tar` archive. Edit your `cassrv-tarball.sh` script and update/replace its contents so that it looks something like this:

```
#!/bin/sh

WORKSPACE=/opt/workspace
SERVER=${WORKSPACE}/cas-overlay-template
WEBAPP=${WORKSPACE}/cas-management-overlay

tar czf /tmp/cassrv-files.tgz --owner=root --group=tomcat --mod
e=g-w,o-rwx \
    -C ${SERVER} etc/cas \
    -C ${SERVER}/target cas --exclude cas/META-INF \
    -C ${WEBAPP} etc/cas \
    -C ${WEBAPP}/target cas-management --exclude cas-management/MET
A-INF

echo ""

ls -asl /tmp/cassrv-files.tgz
exit 0
```

Update the installation shell script

When we created the original `cassrv-tarball.sh` script, we also created a `cassrv-install.sh` script to manage shutting down Tomcat, deleting the old contents of `/etc/cas`, deleting the old copy of the CAS server application (and any associated runtime files), extracting a new copy of the application from the `tar`

archive, and restarting Tomcat. That script can also be extended to handle both the CAS server and the management webapp. Edit your `cassrv-install.sh` script and update/replace its contents so that it looks something like this:

```
#!/bin/sh

echo "--- Installing on `hostname`"
umask 027

if [ -f /tmp/cassrv-files.tgz ]
then
    systemctl stop tomcat

    cd /

    # Only delete/replace etc/cas/services if management webapp is
    # not already installed
    if [ ! -d /opt/tomcat/latest/webapps/cas-management ]
    then
        rm -rf etc/cas/config etc/cas/services
        tar xzf /tmp/cassrv-files.tgz etc/cas
    else
        rm -rf etc/cas/config
        tar xzf /tmp/cassrv-files.tgz etc/cas --exclude etc/cas/servi
ces
    fi

    chmod -fR g+w etc/cas/services
    chmod -f g+w etc/cas/saml

    cd /opt/tomcat/latest/
    rm -rf webapps/cas work/Catalina/localhost/cas
    rm -rf webapps/cas-management work/Catalina/localhost/cas-managem
ent

    cd /opt/tomcat/latest/webapps
    tar xzf /tmp/cassrv-files.tgz cas cas-management

    systemctl start tomcat

    rm -f /tmp/cassrv-files.tgz /tmp/cassrv-install.sh
    echo "Installation complete."
else
    echo "Cannot find /tmp/cassrv-files.tgz; nothing installed."
    exit 1
fi

exit 0
```

Install and test on the master build server

Use the new scripts created above (or repeat the commands) to install the management webapp and updated CAS server files on the master build server (***casdev-master***) and restart Tomcat:

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything seems to be running correctly on the master build server, it can be copied to the CAS servers, again using the scripts above:

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Shut down all but one of the pool servers

Operating CAS with a pool of servers instead of a single server requires special configuration. Because that configuration hasn't been completed yet, testing must be performed against a single server. Therefore, the other servers in the pool should be shut down so that the load balancer will direct all traffic to that single server. Run the command

```
# systemctl stop tomcat
```

on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

Access the webapp

Open up a web browser (in “incognito” or “private browsing” mode) and enter the URL of the management webapp:

```
https://casdev.newschool.edu/cas-management
```

and authenticate as a user listed in the `/etc/cas/config/admusers.properties` file. The default screen of the management webapp, which shows a list of all configured services sorted by order of evaluation, should appear and look something like Figure 22.

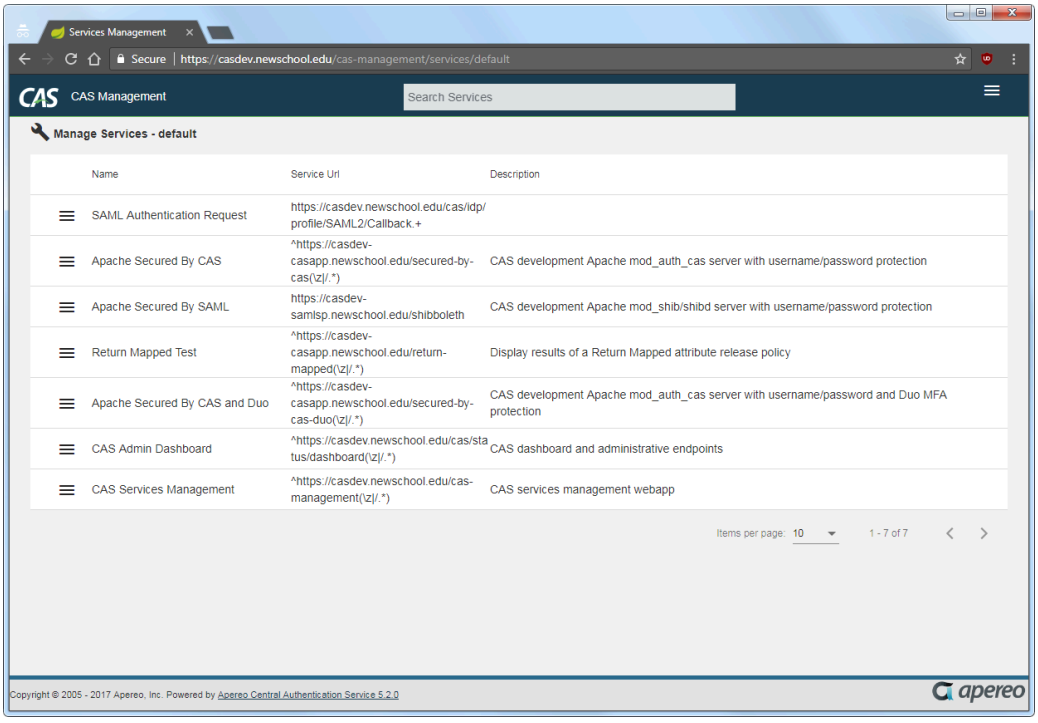

Figure 22. The managemement webapp

Try editing a service registry entry

Select “Edit” from the menu at the left of one of the service definitions (it doesn’t matter which one), and then browse the various tabs across the top of the window to see how all the various aspects of the service definition can be viewed and edited in the webapp. Then select the “Contacts” tab, and try updating the contact information for the service, as shown in Figure 23.

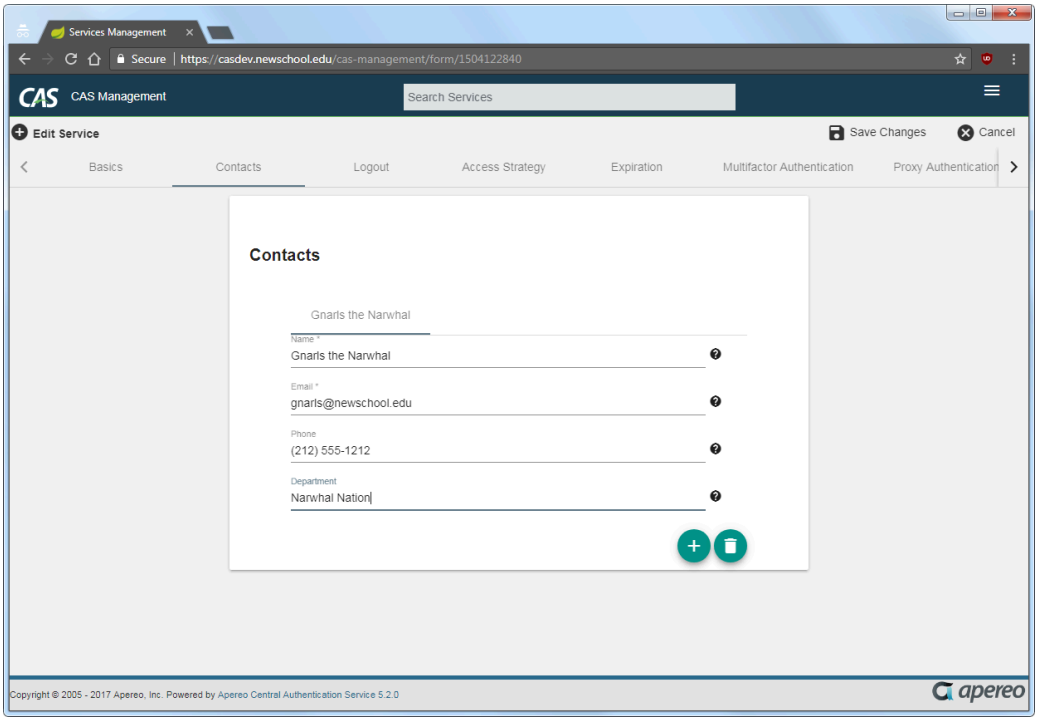

Figure 23. Editing a service definition

Click the “Save Changes” button at the top right of the window, and verify that the changes can be successfully saved (a message will appear at the bottom of the window). Check the log file (`/var/log/cas/cas.log`) and the contents of the changed service definition in `/etc/cas/services`, too.

> **Note:** The management webapp always writes the complete service definition to the registry, not just the parts that are different than the default values. So don’t be surprised when the definition you just edited has a lot more “stuff” in it that it did before you changed it.

Try creating a service registry entry

Select “Add New Service” from the menu at the top right of the page and fill in the “Basics” screen (the values don’t really matter). Then click “Save Changes” and verify that a new service registry definition was successfully created. Check the log file and `/etc/cas/services`, too.

Back on the default page (list of services), select “Delete” from the menu at the left of the service definition you just created, and confirm the deletion. Then verify that the service was indeed deleted.

Restart the pool servers

One testing is complete, run the command

```
# systemctl start tomcat
```

on each of the pool servers shut down previously.

Commit changes to Git

Before moving on to the next task, commit the new service registry definition file in the `cas-overlay-template` directory, as well as all the changes in the `cas-management-overlay` directory (`pom.xml` , `etc/cas/config/management.properties` , and `etc/cas/config/log4j2-management.properties`), to Git to make changes easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/services/CASServiceManagement-1510002272.json
casdev-master# git commit -m "Added service definition for the management webapp"
[newschool-casdev 5581373] Added service definition for the managemen
t webapp
 1 file changed, 8 insertions(+)
 create mode 100644 etc/cas/services/CASServiceManagement-151000227
2.json
casdev-master# cd ../cas-management-overlay
casdev-master# git add pom.xml
casdev-master# git add etc/cas/config/management.properties
casdev-master# git add etc/cas/config/log4j2-management.xml
casdev-master# git commit -m "Enabled the management webapp"
[newschool-casdev-sm 7688bb1] Enabled the management webapp
 4 files changed, 42 insertions(+), 21 deletions(-)
 rewrite etc/cas/config/management.properties (99%)
 delete mode 100644 etc/cas/config/users.properties
casdev-master#
```

on the master build server (***casdev-master***).

Customizing the CAS user interface

Summary: The CAS login page is the first thing users see when logging into any CAS-ified service. It should reflect the branding and style of the organization to be clearly recognizable, and should also take advantage of available features to prevent spoofing attempts.

Branding is important to The New School. As something that every member of our community encounters on a daily basis, the look and feel of the CAS login page is an important representation of that brand. Since 2014, the CAS 3.5.*x* login page has looked like the one shown in Figure 24. There are 15-20 background images showing various New School locations and activities; a different image is randomly selected each time the page is loaded.

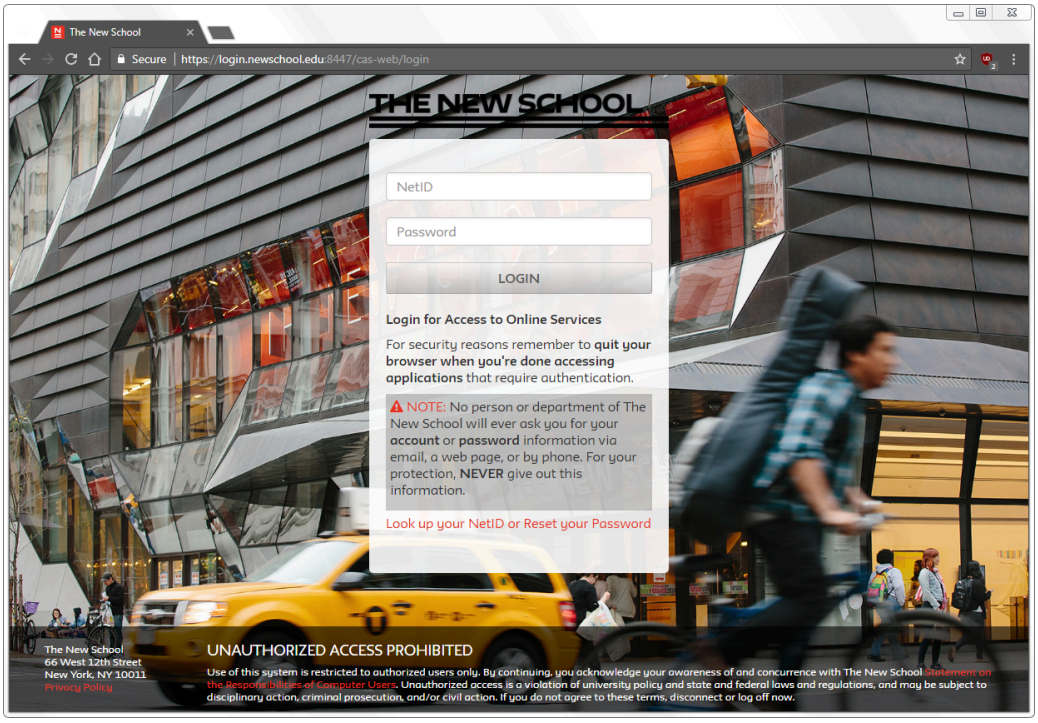

Figure 24. The New School CAS 3.5 login page

After almost four years the page is looking a little dated, the amount of text under the login button has grown over time to be somewhat unattractive, and although it works on mobile devices, it's not as nice an experience as it could be. As part of the upgrade from CAS 3.5.*x* to CAS 5, we will be implementing a brand new login page. The new page will be more in line with current web design practices, will be fully responsive with a mobile-first design, and will also better tie into the current branding on other New School web sites.

Start with a mock-up

The CAS server uses a collection of HTML files, Thymeleaf templates, CSS and JavaScript files, and Java message bundles to implement the login page (and other pages). Rather than try to work with all of these components while designing the login page's look and feel, a process that required several rounds of reviews and changes, we created a simple mock-up of the page using plain old HTML, CSS, and JavaScript and hosted it independently of the CAS server. By doing the design outside the CAS server, we were able to focus on the look and feel of the page itself, rather than the underlying technologies that would eventually be used implement it.

The mock-up that we eventually arrived at is shown in Figure 25. Since CAS 5 uses Bootstrap as its base HTML / CSS / JavaScript framework, we elected to develop our page using that framework as well, which gives us a responsive, mobile-first design. We augmented the base Bootstrap framework with Federico Zivolo's Material Design for Bootstrap theme, which applies Material Design principles to Bootstrap's various elements. The New School's custom *Neue* font and CSS color settings bring the New School branding to the page.

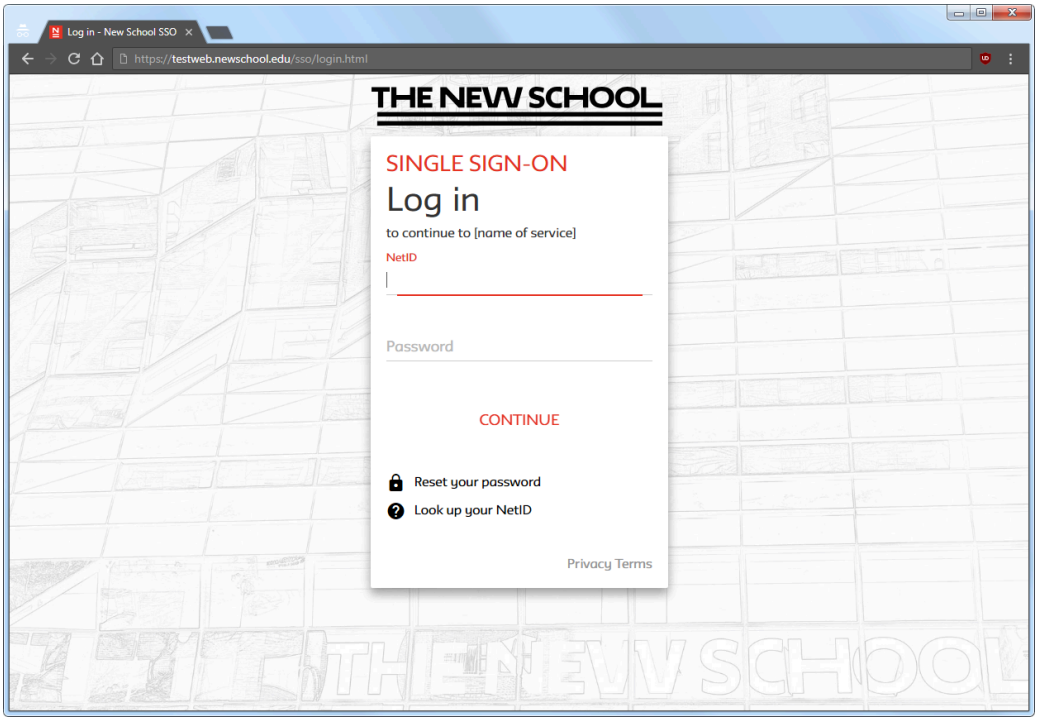

Figure 25. Mock-up of the new login page

The HTML for the mock-up page is actually pretty simple, and looks like this:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
  <meta name="viewport" content="width=device-width, initial-scal
e=1"/>

  <title>Log in - New School SSO</title>

  <!-- JQuery -->
  <script src="//code.jquery.com/jquery-1.10.2.min.js"></script>

  <!-- Bootstrap -->
  <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/
3.3.7/css/bootstrap.min.css">
  <script src="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstra
p.min.js">
    </script>

  <!-- Material Design for Bootstrap -->
  <link rel="stylesheet" href="css/bootstrap-material-design.min.cs
s">
  <link rel="stylesheet" href="css/ripples.min.css">
  <script src="js/material.min.js"></script>
  <script src="js/ripples.min.js"></script>

  <!-- TNS icons -->
  <link rel="icon" type="image/x-icon" href="//www.newschool.edu/favi
con.ico">
  <link rel="apple-touch-icon" href="//www.newschool.edu/framework/im
gs/tns-appletouch-icon.png">

  <!-- Material Design fonts -->
  <link rel="stylesheet" href="//fonts.googleapis.com/css?family=Robo
to:300,400,500,700">
  <link rel="stylesheet" href="//fonts.googleapis.com/icon?family=Mat
erial+Icons">

  <!-- Page-specifc styles -->
  <link rel="stylesheet" href="css/tnsfonts.css">
  <link rel="stylesheet" href="css/newschool.css">
</head>

<body>
  <div class="container-fluid">
    <!-- New School header logo -->
```

```
    <header role="banner" class="tns-header">
      <section class="tns-header-region">
        <div class="tns-lockup">
          <div class="tns-banner text-center">
            <h1 class="tns-uname">
              <!-- Generator: Adobe Illustrator 18.1.0, SVG Export Pl
ug-In.
                   SVG Version: 6.00 Build 0) --><svg xmlns="http://w
ww.w3.org/2000/svg" xml:space="preserve"
                   version="1.1" x="0px" y="0px" viewBox="0 0 224.4 3
0.2"
                   enable-background="new 0 0 224.4 30.2"
                   id="tns-logo-svg">
                <g id="Layer_1">
                  <g>
                    <rect x="4.4" y="19.7" width="220" height="3.5"/>
                    <rect x="4.4" y="26.7" width="220" height="3.5"/>
                    <g>
                      <path d="M9,3.9v12.4H5V3.9H0V0.3h14v3.6H9z"/>
                      <path d="M29.7,10.2H19.2v6.1h-4v-16h4v6.1h10.5V
0.3h4v16h-4V10.2z"/>
                      <path d="M35.5,16.3v-16h11.4v3.6h-7.4v2.7h6V10
h-6v2.7h7.4v3.6H35.5z"/>
                      <path d="M57.7,6.1h-0.6v10.2h-4v-16h4.7l9.3,10h
0.6v-10h4v16h-4.5L57.7,6.1z"/>
                      <path d="M73.5,16.3v-16h11.4v3.6h-7.4v2.7h6V10
h-6v2.7h7.4v3.6H73.5z"/>
                      <path d="M85.2,0.3h4.5l3.5,12.7h0.9l3.2-12.7h
7.5l3.6,12.7h0.9l3-12.7h4.5l-4.4,16h-7.1l-3.7-12.7h-1.1l-3.4,12.7
h-7.1L85.2,0.3z"/>
                      <path d="M121.5,5.2c0-3.4,2.7-5.2,6.2-5.2c
2,0,4.2,0.5,5.5,1.2l-0.8,3.4c-1.3-0.8-3.3-1.2-4.9-1.2
c-1.2,0-2.1,0.5-2.1,1.3c0,2.6,8.2,1.1,8.2,6.5c0,2.9-2,5.4-6.3,5.4
c-1.8,0-3.9-0.2-5.5-1L122,12c1.7,0.8,3.5,1.3,5.5,1.3c
1.8,0,2.2-0.6,2.2-1.4C129.7,9.5,121.5,10.9,121.5,5.2z"/>
                      <path d="M146.9,16.1c-1.3,0.5-2.4,0.5-4.2,0.5
c-5,0-8.3-3.9-8.3-8.3c0-4.5,3.5-8.3,8.7-8.3c1.3,0,2.8,0,4.1,0.5
l-0.4,3.6c-1.4-0.5-2.4-0.5-3.6-0.5c-3,0-4.7,1.7-4.7,4.7c
0,3,2,4.7,4.8,4.7c1.8,0,2.4-0.2,3.6-0.6V16.1z"/>
                      <path d="M163.2,10.2h-10.5v6.1h-4v-16h4v6.1h1
0.5V0.3h4v16h-4V10.2z"/>
                      <path d="M177.8,0c5.5,0,9.5,2.8,9.5,8.2c
0,5.6-4.3,8.4-9.5,8.4c-5.2,0-9.5-2.8-9.5-8.4C168.2,3,172.1,0,177.8,0
z M177.7,13.3c3.1,0,5.4-1.6,5.4-5.1c0-3.2-2.3-4.8-5.4-4.8
s-5.4,1.6-5.4,4.9C172.4,11.7,174.6,13.3,177.7,13.3z"/>
                      <path d="M197.5,0c5.5,0,9.5,2.8,9.5,8.2c
0,5.6-4.3,8.4-9.5,8.4c-5.2,0-9.5-2.8-9.5-8.4C187.9,3,191.8,0,197.5,0
```

```
z M197.4,13.3c3.1,0,5.4-1.6,5.4-5.1c0-3.2-2.3-4.8-5.4-4.8
s-5.4,1.6-5.4,4.9C192.1,11.7,194.3,13.3,197.4,13.3z"/>
                    <path d="M208,0.3h4v12.2h12.4v3.8H208V0.3z"/>
                  </g>
                </g>
              </g>
            </svg>
          </h1> <!-- tns-uname -->
        </div> <!-- tns-banner -->
      </div> <!-- tns-lockup -->
    </section> <!-- tns-header-region -->
  </header> <!-- tns-header -->
</div> <!-- container-fluid -->

<!-- login form box -->
<div id="container" class="container">
  <div id="content">
    <div class="row">
      <div class="col-sm-12 col-md-4 col-md-offset-4">
        <div class="well" id="login">
          <header class="tns-header">
            <div class="tns-banner">
              <h2 class="tns-sitename">Single Sign-On</h2>
            </div>
          </header>

          <h1>Log in</h1>
          <p>to continue to [name of service]</p>

          <form>
            <!-- inputs -->
            <div class="form-group label-floating is-empty">
              <label class="control-label" for="username">NetID</la
bel>
              <input class="form-control" id="username" name="usern
ame"
                type="text" style="cursor: auto;">
            </div>
            <div class="form-group label-floating is-empty">
              <label class="control-label" for="password">Passwor
d</label>
              <input class="form-control" id="password" name="passw
ord"
                type="password" style="cursor: auto;">
            </div>

            <!-- button -->
```

```
              <div class="form-group text-center">
                <button class="btn btn-primary btn-lg" name="submit">
                  Continue
                  <div class="ripple-container"></div>
                </button>
              </div>

              <!-- account help -->
              <div class="form-group">
                <p class="acctopts">
                  <a href="https://account.newschool.edu/cgi-bin/acct
services.pl?f=i&s=pr">
                    <i class="material-icons acctopts">lock</i>
                    Reset your password
                  </a>
                </p>
                <p class="acctopts">
                  <a href="https://account.newschool.edu/cgi-bin/acct
services.pl?f=i&s=gn">
                    <i class="material-icons acctopts">help</i>
                    Look up your NetID
                  </a>
                </p>
              </div>

              <!-- privacy and terms -->
              <div class="form-row text-right privacy-terms">
                <a href="//www.newschool.edu/privacy-policy/" targe
t="_blank">
                  <span>Privacy</span>
                </a>
                <a href="//it.newschool.edu/sites/default/files/uploa
ds/documents/Statement%20on%20the%20Responsibilities%20of%20Compute
r%20Users%20v2.1.pdf" target="_blank">
                  <span>Terms</span>
                </a>
              </div> <!-- form-row -->
            </form>
          </div> <!-- well -->
        </div> <!-- col -->
      </div> <!-- row -->
    </div> <!-- content -->
  </div> <!-- container -->

  <script>
    $(function () {
      $.material.init();
```

```
    });
  </script>
</body>
</html>
```

The CSS styling for all the elements is included in the `newschool.css` file, and the *Neue* font is loaded by `tnsfonts.css`. The script at the bottom of the page initializes the Material Design for Bootstrap theme.

With the design finalized, the next step is to “port” the mock-up into the CAS user interface framework. This is described in the following pages.

References

- Bootstrap
- Material Design
- Material Design for Bootstrap

How CAS themes work

CAS uses Spring Web Flow to perform its processing of login and logout requests (as well as “add-ons” like multi-factor authentication). It’s not necessary to understand the intricacies of Spring Web Flow to customize the CAS user interface, but it does help to have a basic understanding of what’s going on behind the scenes.

A *flow* encapsulates a reusable sequence of steps that guide a user through the completion of some task, such as logging in or logging out (the two main flows provided by CAS). Flows can span multiple HTTP requests, have state, deal with transactional data, are reusable, and may be dynamic and long-running. Each step in a flow is called a *state*. There are five kinds of state:

1. A *view state* uses a dynamically-generated web page, or *view*, to display information to the user or prompt the user for input. Transition to another state is triggered by an event on the web page, such as the user clicking a “submit” button.
2. An *action state* calls a CAS server function and then transitions to another state in the flow depending on the outcome of the function call.
3. A *decision state* evaluates a Boolean expression and then transitions to one of two other states in the flow depending on whether the expression is true or false.
4. A *subflow state* allows one flow to call another flow to perform some steps. Data may be passed between the calling flow and the subflow.
5. An *end state* defines the end of a flow. If the end state is part of the root flow, execution ends. If it is part of a subflow, the calling flow is resumed.

The CAS user interface—the login page, logout page, and other pages displayed for multi-factor authentication and so on—is part of the view state. The other states comprise the “business logic” that determines which page(s) (view state(s)) are ultimately shown to the user.

Parts of the user interface

There are three “parts” to the CAS user interface:

1. **Views.** A set of HTML files, one per view state, as described above. View pages are processed through the Thymeleaf template engine enabling them to dynamically access property settings and variables and evaluate logical expressions using their values. Thymeleaf also makes it possible to define a common layout template (page background, header and footer, navigation elements, style sheet inclusions, etc.) to be shared by all the

view pages.

2. **Themes.** A collection of cascading style sheets, JavaScript files, and image files that are included by the view pages. The CSS files control the colors, fonts, and other style-related aspects of the interface. The JavaScript files define procedures to be executed in the user's browser to handle things like caps-lock detection, multi-factor authentication, etc. The image files include logos, buttons, lines, etc. used by the style sheets.
3. **Message bundles.** Java property files (`messages.properties` for English, `messages_xx.properties` for locale `xx`) that define all the text messages displayed in the various views. The Thymeleaf template engine automatically inserts messages from the proper message bundle (based on the locale setting) into the HTML views through property reference substitution.

The default views, default theme, and default message bundles are automatically included in the CAS WAR file by the Maven build process. The message bundles reside under `WEB-INF/classes`, the default theme resides under `WEB-INF/classes/static` (in `css`, `js`, and `images` subdirectories), and the default views reside under `WEB-INF/classes/templates`.

The default theme itself is defined by the `WEB-INF/classes/cas-default-theme.properties` file:

```
standard.custom.css.file=/css/cas.css
admin.custom.css.file=/css/admin.css
cas.javascript.file=/js/cas.js
```

These properties define the main CSS file for the user views, the main CSS file for the dashboard (admin pages), and the main JavaScript file for the user views, respectively. The paths are rooted at `WEB-INF/classes/static` in the WAR file, which serves as the document root for the CAS web application. This means, for example, that the complete URL for `/css/cas.css` is `https://casdev.newschool.edu/cas/css/cas.css`, which, if accessed, will retrieve the file `/var/lib/tomcat/webapps/cas/WEB-INF/classes/static/css/cas.css`.

Modifying the user interface

When modifying the CAS user interface, there are two options: change the decorative elements (styles and scripts) of the user interface but keep the same structural elements (HTML views), or change both the decorative elements and the structural elements.

Changing decorative elements (styles and scripts)

To define a new theme that will re-style the default views, a `WEB-INF/classes/[theme_name].properties` file that specifies the location of the main CSS and JavaScript files for the theme should be created:

```
standard.custom.css.file:   /themes/[theme_name]/css/cas.css
cas.javascript.file:        /themes/[theme_name]/js/cas.js
admin.custom.css.file:      /themes/[theme-name]/css/admin.css
```

Next, a `WEB-INF/classes/static/themes/[theme_name]` directory should be created, and theme-specific `cas.css`, `admin.css`, and `cas.js` files placed in the appropriate subdirectories (`css` and `js`). Additional subdirectories for `images` and `fonts` can be created here too, if needed.

☑ **Tip:** When you define a custom theme, the CSS and JavaScript files you declare in the `[theme_name].properties` file will be included *instead of* the CSS and JavaScript files from the default theme. If your intention is to keep most of the styling and scripting the same and make only make decorative changes to the default theme (e.g., changing the background color or font size), you may wish to begin with copies of the default files (`WEB-INF/classes/static/css/cas.css` and `WEB-INF/classes/static/js/cas.js`) and add your customizations to them rather than starting with empty files.

Changing structural elements (HTML views)

By default, the new theme created above will be applied to the default views. However, it's also possible to define a new set of views, different from the defaults. To do this, the `[theme_name].properties` file and the locations for the decorative components should be created as described above, and then a `WEB-INF/classes/templates/[theme_name]` directory should be created to hold the HTML view files for the new theme. There is no property setting to inform the server that it should use the new set of views; it will simply use the files in the `WEB-INF/classes/templates/[theme_name]` directory if it exists, or the default files (in `WEB-INF/classes/templates`) if it does not.

🛈 **Note:** If the `WEB-INF/classes/templates/[theme_name]` directory exists, the server will expect to find *all* the views in that directory. Even if you're only

planning to change one or two of the views (e.g., the login and logout pages), you must provide an HTML file for every view that the configured web flows might need. The easiest way to accomplish this is to copy the entire contents of the default template directory (all files and subdirectories) into your custom template directory and then modify the files for the views you wish to customize, rather than starting with an empty directory.

Changing message strings

By default, Thymeleaf will automatically retrieve message strings (using property names) from `WEB-INF/classes/messages.properties` if the English (`en`) locale is in effect, or from `WEB-INF/classes/messages_xx.properties` if another (`xx`) locale is in effect. To change the value of a particular message string, the property that defines that string can be re-defined in `WEB-INF/classes/custom_messages.properties`. New properties can also be defined in this file and used in the HTML views.

Summary

The table below summarizes the various files and directories associated with customizing the CAS user interface. The procedures for actually creating them and including them in the CAS WAR file are described in the following sections.

	CAS default theme	Custom theme
Theme definition	`WEB-INF/classes/cas-default-theme.properties`	`WEB-INF/classes/[theme_name].properties`
Style sheets	`WEB-INF/classes/static/css/*`	`WEB-INF/classes/static/themes/[theme_name]/css/*`
JavaScript	`WEB-INF/classes/static/js/*`	`WEB-INF/classes/static/themes/[theme_name]/js/*`
Images	`WEB-INF/classes/static/images/*`	`WEB-INF/classes/static/themes/[theme_name]/images/*`
HTML Views	`WEB-INF/classes/templates/*`	`WEB-INF/classes/templates/[theme_name]/*`

	CAS default theme	Custom theme

Messages	`WEB-INF/classes/`
	`messages.properties`
	`messages_xx.properties`
		`WEB-INF/classes/`
		`custom_messages.properties`

References

- CAS 5: User Interface Customization: Dynamic Themes
- CAS 5: User Interface Customization: CSS/JavaScript
- CAS 5: User Interface Customization: Views
- CAS Community: “CAS 5.1.x Custom template. Anyone get this working?”
- CAS Community: “CAS 5.2.0-RC1 Theme not resolved correctly”

How Thymeleaf layouts work

As described in the previous section (page 263), CAS displays a dynamically-created web page, or *view*, whenever the web flow enters a view state. CAS 5 uses Thymeleaf, a server-side template engine, to manage the generation of this dynamic content (CAS 3 used Java Server Pages for this purpose).

Thymeleaf uses *processors* to make dynamic changes to a document. A processor may perform variable (or property) substitutions, evaluate expressions, implement conditional statements, or pass information to/from application methods. In HTML documents, processors take the form of specially-defined HTML attributes that are evaluated by the Thymeleaf engine. A set of processors—plus some extra artifacts— is called a *dialect*. Thymeleaf comes with two dialects out of the box: the *Standard Dialect*, which defines a base set of features that should be more than enough for most scenarios, and the *SpringStandard Dialect*, which extends the Standard Dialect with some specific features that integrate Thymeleaf with Spring MVC applications.

CAS makes use of the SpringStandard Dialect and another dialect, called the *Thymeleaf Layout Dialect*.

Thymeleaf Layout Dialect

The Thymeleaf Layout Dialect is a dialect (feature set) for Thymeleaf that allows layouts and reusable templates to be built in order to improve code reuse. The dialect makes use of two major components: layouts and content templates.

Layouts

A *layout* is an HTML file that defines web page components that will be common to all web pages using the layout: the header, footer, menu, navigation, etc. A very simple layout might look like this:

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
  <title>My simple layout</title>
  <link rel="stylesheet" href="common-styles.css"/>
  <script src="common-script.js"></script>
</head>
<body>
  <header>
    <h1>My Little Website</h1>
  </header>
  <section layout:fragment="content">
    <p>Default page content</p>
  </section>
  <footer>
    <p>My little footer</p>
    <p layout:fragment="custom-footer">Default footer text</p>
  </footer>
</body>
</html>
```

The interesting parts of this layout are the `layout:fragment` attribute used on the `<section>` element (line 12) and the `<p>` element in the footer (line 17). These elements are candidates to be replaced by matching fragments from content templates.

Content templates

Content templates provide content to be substituted into a layout. A simple content template might look something like this:

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" layou
t:decorate="~{layout}">
<head>
  <title>Some simple content</title>
  <link rel="stylesheet" href="content-specific-styles.css"/>
  <script src="content-specific-script.js"></script>
</head>
<body>
  <p>Here is some text from the content template.</p>
  <section layout:fragment="content">
    <p>This is a paragraph from the content template.</p>
  </section>
  <footer>
    <p layout:fragment="custom-footer">This is the custom footer fro
m the content template.</p>
  </footer>
</body>
</html>
```

The `layout:decorate` attribute on the `<html>` tag is required, and tells Thymeleaf which layout will be “decorated” using this content template. The value in the curly braces is the name of the layout; Thymeleaf will look for a file with that name and a `.html` suffix to find the file containing the layout. The content template above defines its own title, style sheet, and script file; it also defines *fragments* named `content` and `custom-footer`.

After Thymeleaf processes the content template, the resulting page sent to the user’s browser will be:

```
<!DOCTYPE html>
<html>
<head>
  <title>Some simple content</title>
  <link rel="stylesheet" href="common-styles.css"/>
  <script src="common-script.js"></script>
  <link rel="stylesheet" href="content-specific-styles.css"/>
  <script src="content-specific-script.js"></script>
</head>
<body>
  <header>
    <h1>My Little Website</h1>
  </header>
  <section>
    <p>This is a paragraph from the content template.</p>
  </section>
  <footer>
    <p>My little footer</p>
    <p>This is the custom footer from the content template.</p>
  </footer>
</body>
</html>
```

The content template decorated `layout.html`, resulting in a combination of the layout plus the fragments of the content template:

- The body of the content template's `<head>` element has been appended to the body of the layout's `<head>` element to produce a single, merged `<head>` element.
- The body of the content template's `<title>` element has replaced (overridden) the body of the layout's `<title>` element.
- The elements in the layout with a `layout:fragment` attribute (the `<section>` and `<p>` elements mentioned earlier) have been replaced with the contents of the corresponding fragments from the content template.
- Anything *outside* of a `layout:fragment` in the `<body>` of the content template (e.g., "Here is some text from the content template") has been discarded.
- All other elements in the layout have been reproduced verbatim.

Content templates don't have to provide definitions for all fragments referenced by the layout template. For example:

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" layou
t:decorate="~{layout}">
<head>
<body>
  <section layout:fragment="content">
    <p>This is a paragraph from the content template.</p>
  </section>
</body>
</html>
```

would result in:

```
<!DOCTYPE html>
<html>
<head>
  <title>My simple layout</title>
  <link rel="stylesheet" href="common-styles.css"/>
  <script src="common-script.js"></script>
</head>
<body>
  <header>
    <h1>My Little Website</h1>
  </header>
  <section>
    <p>This is a paragraph from the content template.</p>
  </section>
  <footer>
    <p>My little footer</p>
    <p>Default footer text</p>
  </footer>
</body>
</html>
```

In this case, the page title is the one defined in the layout, because the content template did not define one. And the text from the layout also appears in the page footer, because the content template did not define a `custom-footer` fragment.

The `layout:title-pattern` attribute

As shown above, if the content template defines a `<title>` element, the content of that element will replace the content of a `<title>` element defined in the layout template. But for occasions in which the page title includes both a “standard” part and a “variable” part, Thymeleaf provides the `layout:title-pattern` attribute for use in the layout template. A layout that contains

```
<title layout:title-pattern="$LAYOUT_TITLE - $CONTENT-TITLE">Roses ar
e red</title>
```

and a content template that contains

```
<title>Violets are blue</title>
```

would result in

```
<title>Roses are red - Violets are blue</title>
```

Alternatively, a layout contains

```
<title layout:title-pattern="$CONTENT_TITLE - $LAYOUT-TITLE">Roses ar
e red</title>
```

and the same content template would result in

```
<title>Violets are blue - Roses are red</title>
```

Thymeleaf SpringStandard Dialect

As shown above the Thymeleaf Layout Dialect defines some special HTML attributes (`layout:decorate` , `layout:fragment` , and `layout:title-pattern` , among others) that allow templates to be processed and turned into web pages. The rest of the Thymeleaf functionality used by CAS is provided by the

SpringStandard Dialect (which, as mentioned above, is an extension of the Standard Dialect). The dialect features most commonly used in CAS views are described in the following subsections; for a complete description of all features, consult the documentation linked at the bottom of this page.

Value substitutions

Thymeleaf allows values to be retrieved from Java properties, methods, classes, variables, etc.

`#themes.code('property.name')`

Retrieve the value of `property.name` from the `[theme_name].properties` file and substitute it here.

`#{message.key}`

Look up the value of `message.key` in the `messages.properties` (or `messages_xx.properties`) bundle, or the overriding `custom_messages.properties` bundle, and substitute it here.

`${expression}`

Evaluate the expression, written in Spring Expression Language, and substitute the result here. The expression may contain, among other things, literals, boolean and relational operators, regular expressions, class expressions, method invocations, variables, bean references, access to properties, etc. Usually, in CAS templates, the expression is either a simple variable reference or method call to obtain a value.

`@{url}`

Treat `url` (which may be an expression) as a URL. This construct is usually used in conjunction with one of the attribute modifiers below.

Attribute modifiers

Thymeleaf allows the values for attributes of HTML elements to be computed dynamically at runtime and “injected” into the element. To do this, special Thymeleaf variants of the attributes are used in place of the regular attributes. For example:

```
<a th:href="@{#{screen.welcome.privacy.url}}">Privacy Policy</a>
```

will look up the value of `screen.welcome.privacy.url` in `messages.properties` and then set the `href` attribute of the `<a>` tag to this URL.

There are several dozen attributes for which Thymeleaf will perform this service, but the most commonly used ones in the CAS views are `th:href` and `th:src` for URL attributes (mostly used on `<a>`, `<img>`, `<script>`, and `<link>` tags), and `th:value` for HTML form input element default values.

Text (tag body) modifiers

Thymeleaf also allows the values (contents) of HTML elements themselves to be computed dynamically at runtime. For example:

```
<p th:text="#{home.welcome}">Welcome to our widget store!</p>
```

will look up the value of `home.welcome` in `messages.properties` and then set the contents of the `<p>` element to that value, replacing the words “Welcome to our widget store!”.

The `th:text` modifier substitutes “escaped” text into an element, while the `th:utext` modifier substitutes “unescaped” text. The difference can be seen when the string to be substituted contains HTML tags or entity references instead of simple plain text. For example, if `messages.properties` contains

```
home.welcome=Welcome to our <b>fantastic</b> widget store!
```

then the example above will result in

```
<p>Welcome to our &lt;b&gt;fantastic&lt;/b&gt; widget store!</p>
```

which is probably not what was intended. On the other hand, this:

```
<p th:utext="#{home.welcome}">Welcome to our widget store!</p>
```

will produce

```
<p>Welcome to our <b>fantastic</b> widget store!</p>
```

which is the intended result.

Conditional evaluation

The `th:if` and `th:unless` attributes will evaluate an expression and, depending on the result, cause the element containing the attribute to be included in the output (what's sent to the browser) or not. The `th:if` attribute will cause the element to be included if the expression evaluates to true; the `th:unless` attribute will cause the element to be included if the expression evaluates to false. For example, the CAS login form contains this:

```
<div class="registered-service" th:if="${registeredService}">
  <div th:if="${serviceUIMetadata}">
    <p>to continue to
       <span th:text="${serviceUIMetadata.displayName}"></span></p>
  </div>
  <div th:unless="${serviceUIMetadata}">
    <p>to continue to
      <span th:text="${registeredService.name}"></span></p>
  </div>
</div>
```

The outermost `<div>` and its contents will be included in the output only if the `registeredService` variable is set and non-null. If the variable is set, then the string `to continue to [service_name]` will be displayed. Depending on whether or not the `serviceUIMetadata` variable is set and non-null (i.e., whether the service is a SAML-based service or not), the value of `[service_name]` will be obtained from the SAML2 metadata or the service registry entry.

Re-usable fragments

Thymeleaf also makes it possible to include fragments from other files, making it possible to define a fragment that can be used in more than one content template. For example, the line

```
<div th:replace="fragments/footer"/>
```

will replace the `<div>` element (the "host tag") with the contents of the file `fragments/footer.html`. The `th:replace` directive may be used in both layout templates and content templates; it may also be used within the fragments themselves (e.g., `content.html` may contain `th:reaplce="fragments/foo"`,

which in turn may contain `th:replace="fragments/bar"`). There is also a `th:insert` directive that will insert the contents of the fragment into the host tag, rather than replacing the host tag.

The CAS templates

The CAS server's default Thymeleaf templates are kept in `WEB-INF/classes/templates`:

```
casdev-master# cd /var/lib/tomcat/cas/WEB-INF/classes/templates
casdev-master# ls -F
casAcceptableUsagePolicyView.html
casAccountDisabledView.html
casAccountLockedView.html
...
casLoginView.html
casLogoutView.html
...
fragments/
layout.html
...
casdev-master#
```

There are a little more than three dozen files whose names begin with `cas`; these are the content templates for the various views. There is one file for each view state defined by the CAS web flows and subflows; the names of the files make it fairly obvious which file belongs to each state.

The `layout.html` file is the layout template for all the views; this is where the main components of the user interface look and feel are defined.

The `fragments` directory contains re-usable HTML code fragments that are included into other templates with `th:replace`.

References

- Thymeleaf Layout Dialect: Examples
- Thymeleaf: Tutorial: Using Thymeleaf
- Thymelead: Tutorial: Thymeleaf + Spring

Add a new theme to the overlay

The New School CAS login page is different enough from the CAS default page that we will need to provide both custom decorative elements (styles and scripts) and custom structural elements (HTML views). The files associated with these new elements will be incorporated into the CAS WAR file using the Maven overlay.

Create the Maven `src` directory

To include original source material in the CAS WAR file created by the Maven build process, that material should be placed in the `src` directory of the Maven project. Maven uses the `src` directory and various subdirectories therein to organize the source material and determine when it should be used and where it should be copied to (if anywhere). The reference documentation, linked at the bottom of this page, provides more detail about this.

For the purposes of including the files associated with our new theme, we will use the `src/main/resources` directory. Maven will copy the contents of this directory into the WAR file under `WEB-INF/classes` (i.e., `src/main/resources/foo` will appear in the WAR file as `WEB-INF/classes/foo`). This is the “overlay” part of our Maven WAR overlay project—files and subdirectories with unique names will be added to the WAR file, while files and subdirectories with conflicting names will replace (overwrite) the originals.

Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# mkdir -p src/main/resources
```

on the master build server (***casdev-master***) to create this directory.

Define the `newschool` theme

To define the `newschool` theme, first create the file `src/main/resources/newschool.properties` (recall that the contents of `src/main/resources` will be overlaid onto `WEB-INF/classes`). This file will specify the decorative elements (styles and scripts) associated with the new theme:

```
standard.custom.css.file:   /themes/newschool/css/newschool.css
admin.custom.css.file:      /themes/newschool/css/admin.css
cas.javascript.file:        /themes/newschool/js/newschool.js
```

The files may be named any way that makes sense; they do not have to be named `cas.css`, `cas.js`, etc. Here we have chosen to use the name of the theme to make it clear when we include them what they are. Next, create the directory structure that will hold the theme's decorative elements (page 265) by running the commands

```
casdev-master# cd src/main/resources
casdev-master# for sub in css js images fonts
> do
> mkdir -p static/themes/newschool/${sub}
> done
casdev-master#
```

These are the subdirectories that will hold the theme's CSS style sheets, JavaScript files, images, and fonts.

Copy in theme files from the mock-up

We already created many of the files that will reside in these subdirectories when we created the mock-up website (see the HTML shown in the overview (page 256)). So, rather than create them again from scratch, we will copy them from there:

```
casdev-master# cd static/themes/newschool
```

Copy the CSS files (we have chosen to merge `newschool.css` and `testweb.css` into a single file):

```
casdev-master# curl -L https://testweb.newschool.edu/sso/css/newschoo
l.css https://testweb.newschool.edu/sso/css/tnsfonts.css > newschoo
l.css
casdev-master# curl -L https://testweb.newschool.edu/sso/css/bootstra
p-material-design.min.css -o css/bootstrap-material-design.min.css
casdev-master# curl -L https://testweb.newschool.edu/sso/css/ripple
s.min.css -o css/ripples.min.css
casdev-master# touch css/admin.css
```

We create an empty `admin.css` file, since it's referenced in `newschool.properties` above but was not part of the mock-up website. Next, copy the JavaScript files:

```
casdev-master# curl -L https://testweb.newschool.edu/sso/js/materia
l.min.js -o js/material.min.js
casdev-master# curl -L https://testweb.newschool.edu/sso/js/ripples.m
in.js -o js/ripples.min.js
casdev-master# touch js/newschool.js
```

We create an empty `newschool.js` file, since it's referenced in `newschool.properties` above but was not part of the mock-up website. Next, copy the image files:

```
casdev-master# curl -L https://testweb.newschool.edu/sso/images/backg
round.jpg -o images/background.jpg
casdev-master# curl -L https://www.newschool.edu/favicon.ico -o image
s/favicon.ico
casdev-master# curl -L https://www.newschool.edu/framework/imgs/tns-a
ppletouch-icon.png -o images/appleicon.png
```

The mock-up website pulled these files from difference sources; we've collected them all into the `images` subdirectory for the CAS server. Finally, copy the font files:

```
casdev-master# wget -q -np -nH -R '*html*' --cut-dirs=1 -r https://te
stweb.newschool.edu/sso/fonts/
```

There are, of course, other ways these files can be copied into the overlay, or they can be created from scratch if there is no mock-up website to work from.

Create the `newschool` template set

Rather than creating an entire new set of HTML views (Thymeleaf templates) from scratch, we well start with a copy of the default views and customize them.

The easiest way to get a copy of the default templates is to simply copy them from the deployed application directory . Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# cd src/main/resources
casdev-master# mkdir templates
casdev-master# cp -rp /var/lib/tomcat/cas/WEB-INF/classes/templates t
emplates/newschool
casdev-master# chown -R root.root templates/newschool
casdev-master# chmod -R og+rX templates/newschool
```

to copy the default templates from the deployed application directory to the directory structure that will hold the theme's structural elements (page 265).

The complete theme directory structure

Once all the commands above have been executed, the `src/main/resources` directory in the overlay should look like this:

```
src
└── main/
    └── resources/
        ├── custom_messages.properties
        ├── newschool.properties
        ├── static/
        │   └── themes/
        │       └── newschool/
        │           ├── css/
        │           │   ├── admin.css
        │           │   ├── bootstrap-material-design.min.css
        │           │   ├── newschool.css
        │           │   └── ripples.min.css
        │           ├── fonts/
        │           │   └── Neue/
        │           │       ├── Neue-Black.eot
        │           │       ├── Neue-Black.svg
        │           │       ├── Neue-Black.ttf
        │           │       ├── Neue-Black.woff
        │           │       ├── Neue-Bold.eot
        │           │       ├── Neue-Bold.svg
        │           │       ├── Neue-Bold.ttf
        │           │       ├── Neue-Bold.woff
        │           │       ├── Neue-BoldItalic.svg
        │           │       ├── Neue-BoldItalic.ttf
        │           │       ├── Neue-BoldItalic.woff
        │           │       ├── Neue-Regular.eot
        │           │       ├── Neue-Regular.svg
        │           │       ├── Neue-Regular.ttf
        │           │       ├── Neue-Regular.woff
        │           │       ├── Neue-RegularItalic.eot
        │           │       ├── Neue-RegularItalic.svg
        │           │       ├── Neue-RegularItalic.ttf
        │           │       ├── Neue-RegularItalic.woff
        │           │       ├── NeueDisplay-Black.eot
        │           │       ├── NeueDisplay-Black.svg
        │           │       ├── NeueDisplay-Black.ttf
        │           │       ├── NeueDisplay-Black.woff
        │           │       ├── NeueDisplay-Random.eot
        │           │       ├── NeueDisplay-Random.svg
        │           │       ├── NeueDisplay-Random.ttf
        │           │       ├── NeueDisplay-Random.woff
        │           │       ├── NeueDisplay-Ultra.eot
        │           │       ├── NeueDisplay-Ultra.svg
        │           │       ├── NeueDisplay-Ultra.ttf
        │           │       ├── NeueDisplay-Ultra.woff
        │           │       ├── NeueDisplay-Wide.eot
```

```
        │           │       ├── NeueDisplay-Wide.svg
        │           │       ├── NeueDisplay-Wide.ttf
        │           │       └── NeueDisplay-Wide.woff
        │           ├── images/
        │           │   ├── appleicon.png
        │           │   ├── background.jpg
        │           │   └── favicon.ico
        │           └── js/
        │               ├── material.min.js
        │               ├── newschool.js
        │               └── ripples.min.js
        └── templates/
            └── newschool/
                ├── casAcceptableUsagePolicyView.html
                ├── casAccountDisabledView.html
                ├── casAccountLockedView.html
                ├── casAuthenticationBlockedView.html
                ├── casAuthyLoginView.html
                ├── casAzureAuthenticatorLoginView.html
                ├── casBadHoursView.html
                ├── casBadWorkstationView.html
                ├── casConfirmLogoutView.html
                ├── casConfirmView.html
                ├── casConsentLogoutView.html
                ├── casConsentReviewView.html
                ├── casConsentView.html
                ├── casDuoLoginView.html
                ├── casExpiredPassView.html
                ├── casGenericSuccessView.html
                ├── casGoogleAuthenticatorLoginView.html
                ├── casGoogleAuthenticatorRegistrationView.html
                ├── casGuaDisplayUserGraphicsView.html
                ├── casGuaGetUserIdView.html
                ├── casInterruptView.html
                ├── casLoginMessageView.html
                ├── casLoginView.html
                ├── casLogoutView.html
                ├── casMfaRegisterDeviceView.html
                ├── casMustChangePassView.html
                ├── casPac4jStopWebflow.html
                ├── casPasswordUpdateSuccessView.html
                ├── casPropagateLogoutView.html
                ├── casRadiusLoginView.html
                ├── casResetPasswordErrorView.html
                ├── casResetPasswordSendInstructionsView.html
                ├── casResetPasswordSentInstructionsView.html
                ├── casResetPasswordVerifyQuestionsView.html
```

```
├── casRiskAuthenticationBlockedView.html
├── casServiceErrorView.html
├── casSurrogateAuthnListView.html
├── casSwivelLoginView.html
├── casU2fLoginView.html
├── casU2fRegistrationView.html
├── casYubiKeyLoginView.html
├── casYubiKeyRegistrationView.html
├── error/
│   ├── 401.html
│   ├── 403.html
│   ├── 404.html
│   ├── 405.html
│   └── 423.html
├── error.html
├── fragments/
│   ├── bottom.html
│   ├── cas-resources-list.html
│   ├── cookies.html
│   ├── defaultauthn.html
│   ├── footer.html
│   ├── footerButtons.html
│   ├── head.html
│   ├── insecure.html
│   ├── loginProviders.html
│   ├── loginform.html
│   ├── loginsidebar.html
│   ├── logo.html
│   ├── modal.html
│   ├── pwdupdateform.html
│   ├── serviceui.html
│   └── top.html
├── layout.html
├── monitoring/
│   ├── attrresolution.html
│   ├── layout.html
│   ├── viewAuthenticationEvents.html
│   ├── viewConfig.html
│   ├── viewConfigMetadata.html
│   ├── viewDashboard.html
│   ├── viewLoggingConfig.html
│   ├── viewSsoSessions.html
│   ├── viewStatistics.html
│   └── viewTrustedDevices.html
└── protocol/
    ├── 2.0/
    │   ├── casProxyFailureView.html
```

```
│   ├── casProxySuccessView.html
│   ├── casServiceValidationFailure.html
│   └── casServiceValidationSuccess.html
├── 3.0/
│   ├── casServiceValidationFailure.html
│   └── casServiceValidationSuccess.html
├── casPostResponseView.html
├── oauth/
│   └── confirm.html
├── oidc/
│   └── confirm.html
└── openid/
    ├── casOpenIdAssociationSuccessView.html
    ├── casOpenIdServiceFailureView.html
    ├── casOpenIdServiceSuccessView.html
    └── user.html
```

The following pages in this section will discuss how to customize these pages.

References

- Apache Maven: Introduction to the Standard Directory Layout

Build and deploy the overlay

Now that the theme and template files have been added to the overlay, the overlay has to be rebuilt and deployed. The customization process then becomes, basically:

1. Edit one or more of the files
2. Check the results in a web browser
3. Rinse and repeat

until happy. Doing this on the master build server and rebuilding/re-deploying the sever after every change is clearly not a very productive way to accomplish this process. Therefore, we will instead edit the files directly on the deployed CAS server, and once we're happy with them, copy the final versions back into the overlay on the master build server.

Configure user interface properties

Edit the file `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and add the following line:

```
spring.thymeleaf.cache:                 false
```

This turns off caching in the Thymeleaf engine, so that when changes are made to the HTML views on the server, they will be immediately re-processed by Thymeleaf and visible when the browser reloads the pages. When caching is enabled, pages are only processed when the engine first loads them, meaning that changes will not be visible until the server is restarted.

> ⚠ **Important:** Disabling Thymeleaf caching may have a negative impact on CAS server performance, especially in high-load environments. Therefore, it should only be disabled in development and/or test environments. On production CAS servers, this property should **always** be set to `true`.

Next, add the following line, which will make all services use the new theme and template files instead of the defaults:

```
cas.theme.defaultThemeName:             newschool
```

> ℹ **Note:** This setting forces all services to use the same theme, which makes sense in most environments. For environments that need (or want) to have different themes applied to different services, it's also possible to set the theme on a per-service basis via the service registry.

Rebuild the server

Run Maven to rebuild the server to include the new theme and view files:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 01:00 min
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 35M/84M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Shut down all but one of the pool servers

Operating CAS with a pool of servers instead of a single server requires special configuration. Because that configuration hasn't been completed yet, testing must be performed against a single server. Further, since we're going to be editing the CSS, JavaScript, and HTML files and looking for those changes, we want to make sure we're accessing the same server (the one where we're editing the files) every time. Therefore, the other servers in the pool should be shut down so that the load balancer will direct all traffic to that single server. Run the command

```
# systemctl stop tomcat
```

on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

References

- CAS 5: Configuration Properties: Themes
- CAS 5: Configuration Properties: Views

Overview

Now that the files for the new theme have been put into the Maven overlay and deployed to the CAS servers, it's time to customize their content. This process can be broken down into four main steps:

1. Identify the "applies to all pages" elements of the HTML in the mock-up login page created earlier, and merge them into the layout template (`layout.html`).
2. Update the login view to use the new layout template, and copy the login page-specific elements of the HTML in the mock-up login page to the files that make up the login view (`casLoginView.html` and `fragments/loginform.html`).
3. Update the logout view to use the new layout template.
4. Update the other views used by the server to use the new layout template.

As mentioned previously, it's easiest to perform this work on the "live" files on one of the deployed CAS servers to avoid the need to rebuild and re-deploy the overlay after every change. Once an acceptable set of files has been created, they can be copied back to the source (overlay template) directory and committed to `git` . For the purposes of the following changes, ***casdev-srv01*** will be that server. Begin by changing to the deployed CAS server directory:

```
casdev-srv01# cd /var/lib/tomcat/cas/WEB-INF/classes
```

Update the layout template

In general, the layout template is responsible for providing most of the `<head>` element content across all the views, so that they all look the same. And it's also responsible for providing the common-to-all-views components of the `<body>` element (headers, footers, and such). The main tasks in updating the layout template, therefore, are merging the `<head>` and `<body>` elements we created in the mock-up login page with the ones provided by the CAS project in `layout.html`.

Merging the `<head>` elements

The mock-up login page includes a number of external style sheets, scripts, and fonts:

```
<head>
  <meta charset="UTF-8"/>
  <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
  <meta name="viewport" content="width=device-width, initial-scal
e=1"/>

  <title>Log in - New School SSO</title>

  <!-- JQuery -->
  <script src="//code.jquery.com/jquery-1.10.2.min.js"></script>

  <!-- Bootstrap -->
  <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/
3.3.7/css/bootstrap.min.css">
  <script src="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstra
p.min.js"></script>

  <!-- Material Design for Bootstrap -->
  <link rel="stylesheet" href="css/bootstrap-material-design.min.cs
s">
  <link rel="stylesheet" href="css/ripples.min.css">
  <script src="js/material.min.js"></script>
  <script src="js/ripples.min.js"></script>

  <!-- TNS icons -->
  <link rel="icon" type="image/x-icon" href="//www.newschool.edu/favi
con.ico">
  <link rel="apple-touch-icon" href="//www.newschool.edu/framework/im
gs/tns-appletouch-icon.png">

  <!-- Material Design fonts -->
  <link rel="stylesheet" href="//fonts.googleapis.com/css?family=Robo
to:300,400,500,700">
  <link rel="stylesheet" href="//fonts.googleapis.com/icon?family=Mat
erial+Icons">

  <!-- Page-specifc styles -->
  <link rel="stylesheet" href="css/tnsfonts.css">
  <link rel="stylesheet" href="css/newschool.css">
</head>
```

The default layout template (`templates/layout.html`) provides some of the same style sheets and scripts (e.g., jQuery, Bootstrap), but it provides them locally through *webjars* instead of lining to content distribution servers on the Internet. It also includes some additional scripts that are used by CAS features (password strength meter, geolocation, etc.):

```
<head>
  <meta charset="UTF-8"/>
  <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
  <meta name="viewport" content="width=device-width, initial-scal
e=1"/>

  <title layout:title-pattern="$CONTENT_TITLE - $LAYOUT_TITLE">CAS
&#8211; Central Authentication Service</title>

  <link rel="stylesheet" th:href="@{#{webjars.fontawesomemin.css}}"/>
  <link type="text/css" rel="stylesheet" th:href="@{#{webjars.bootstr
apmin.css}}"/>
  <link type="text/css" rel="stylesheet" th:href="@{#{webjars.latomi
n.css}}"/>

  <link rel="stylesheet" th:href="@{${#themes.code('standard.custom.c
ss.file')}}"/>

  <link rel="icon" th:href="@{/favicon.ico}" type="image/x-icon"/>

  <script type="text/javascript" th:src="@{#{webjars.zxcvbn.js}}"></s
cript>
  <script type="text/javascript" th:src="@{#{webjars.jquerymin.j
s}}"></script>
  <script type="text/javascript" th:src="@{#{webjars.jqueryui.j
s}}"></script>
  <script type="text/javascript" th:src="@{#{webjars.jquerycookie.j
s}}"></script>
  <script src="//www.google.com/recaptcha/api.js" async defer th:i
f="${recaptchaSiteKey}"></script>
  <script th:src="@{#{webjars.bootstrapmin.js}}"></script>

  <script th:inline="javascript">
    /*<![CDATA[*/

    var trackGeoLocation = /*[[${trackGeoLocation}]]*/ === "true";

    var googleAnalyticsTrackingId = /*[[${googleAnalyticsTrackingI
d}]]*/;

    if (googleAnalyticsTrackingId != null && googleAnalyticsTrackingI
d != '') {
        (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObjec
t']=r;i[r]=i[r]||function(){
                    (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*ne
w Date();a=s.createElement(o),
                m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.pa
```

```
rentNode.insertBefore(a,m)
        })(window, document, 'script', 'https://www.google-analytic
s.com/analytics.js', 'ga');

        ga('create', googleAnalyticsTrackingId, 'auto');
        ga('send', 'pageview');
    }

    /*]]>*/
  </script>
</head>
```

The goal then, is to merge these two `<head>` elements together, along with some other appropriate changes, and put them all together in our customized layout template (`templates/newschool/layout.html`):

```
<head>
  <meta charset="UTF-8"/>
  <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
  <meta name="viewport" content="width=device-width, initial-scal
e=1"/>

  <title layout:title-pattern="$CONTENT_TITLE - $LAYOUT_TITLE">
    New School SSO
  </title>

  <!-- JQuery -->
  <script type="text/javascript" th:src="@{#{webjars.jquerymin.j
s}}"></script>
  <script type="text/javascript" th:src="@{#{webjars.jqueryui.j
s}}"></script>
  <script type="text/javascript" th:src="@{#{webjars.jquerycookie.j
s}}"></script>

  <!-- Bootstrap -->
  <link rel="stylesheet" type="text/css" th:href="@{#{webjars.bootstr
apmin.css}}">
  <script th:src="@{#{webjars.bootstrapmin.js}}"></script>

  <!-- Material Design for Bootstrap -->
  <link rel="stylesheet" th:href="@{${#themes.code('md4b.css.fil
e')}}">
  <link rel="stylesheet" th:href="@{${#themes.code('ripples.css.fil
e')}}">
  <script th:src="@{${#themes.code('md4b.js.file')}}"></script>
  <script th:src="@{${#themes.code('ripples.js.file')}}"></script>

  <!-- Material Design fonts -->
  <link rel="stylesheet" href="//fonts.googleapis.com/css?family=Robo
to:300,400,500,700">
  <link rel="stylesheet" href="//fonts.googleapis.com/icon?family=Mat
erial+Icons">

  <!-- 'newschool' theme -->
  <link rel="stylesheet" th:href="@{${#themes.code('standard.custom.c
ss.file')}}">

  <!-- TNS icons -->
  <link rel="icon" type="image/x-icon" th:href="@{${#themes.code('fav
icon.img.file')}}">
  <link rel="icon" type="image/png" th:href="@{${#themes.code('applei
con.img.file')}}">
```

```
  <script th:if="${recaptchaSiteKey}" async defer
    src="//www.google.com/recaptcha/api.js"></script>

  <script th:inline="javascript">
    /*<![CDATA[*/

    var trackGeoLocation = /*[[${trackGeoLocation}]]*/ === "true";
    var googleAnalyticsTrackingId = /*[[${googleAnalyticsTrackingI
d}]]*/;

    if (googleAnalyticsTrackingId != null && googleAnalyticsTrackingI
d != '') {
      (function(i, s, o, g, r, a, m) {
        i['GoogleAnalyticsObject'] = r;
        i[r] = i[r] || function() {
          (i[r].q = i[r].q || []).push(arguments)
        }, i[r].l = 1 * new Date();
        a = s.createElement(o),
          m = s.getElementsByTagName(o)[0];
        a.async = 1;
        a.src = g;
        m.parentNode.insertBefore(a, m)
      })(window, document, 'script', 'https://www.google-analytics.co
m/analytics.js', 'ga');

      ga('create', googleAnalyticsTrackingId, 'auto');
      ga('send', 'pageview');
    }

    /*]]>*/
  </script>
</head>
```

- **Lines 6-8.** Adopt the “dual title” format from the CAS server (explained in How Thymeleaf layouts work (page 268)), but change the layout component of the title to “New School SSO.”
- **Lines 10-17.** Pull in the jQuery and Bootstrap packages using the CAS-provided webjar URLs instead of the Internet content distribution server URLs we used in the mock-up.
- **Lines 19-24.** Pull in the Material Design for Bootstrap CSS and JavaScript files. But instead of hard-coding the paths to these files here as we did in the mock-up, use Thymeleaf’s `#theme.code()` function to retrieve them from values defined in the theme definition file (`newschool.properties`):

```
md4b.css.file:              /themes/newschool/css/bootstrap-materia
l-design.min.css
md4b.js.file:               /themes/newschool/js/material.min.js
ripples.css.file:           /themes/newschool/css/ripples.min.css
ripples.js.file:            /themes/newschool/js/ripples.min.js
```

- **Lines 26-28.** Pull in the Material Design fonts (used by the Material Design for Bootstrap package) from Google's content distribution servers.
- **Lines 30-35.** Pull in the New School cascading style sheet and the favicons, again by using Thymeleaf's `#theme.code()` function to retrieve the paths from `newschool.properties`:

```
standard.custom.css.file:   /themes/newschool/css/newschool.css

favicon.img.file:           /themes/newschool/images/favicon.ico
appleicon.img.file:         /themes/newschool/images/appleicon.png
```

- **Lines 37-64.** Copy the feature-specific scripts from the CAS-provided `layout.html` to the new one.

> ℹ **Note:** Webjars allow developers to bundle their client-side web libraries (JavaScript, CSS, etc.) as JAR files. They don't change the way the libraries themselves work, they just make it possible to manage them using dependency-management tools such as Maven and Gradle. It makes sense to incorporate the CAS-provided webjar version of a library if one exists to ensure we're getting the version that the CAS project supports. But we don't need to package our own locally-added web libraries (such as Material Design for Bootstrap) this way, because we're not managing them as external dependencies.

Merging the `<body>` elements

The default layout template defines a high-level structure for all the web views. It includes a logo, some content, a footer, and a bottom:

```
<body>
  <div id="container" class="container">
    <div th:replace="fragments/logo"/>
    <div layout:fragment="content" id="content">
      <h1/>
      <p/>
    </div>
    <div th:replace="fragments/footer"/>
  </div>

  <div th:insert="fragments/bottom"/>
</body>
```

The content part of the page will be populated from a content template, while the other parts will be taken from re-usable fragments. Our custom layout will use a similar high-level structure:

```
<body>
  <div class="container-fluid">
    <div th:replace="newschool/fragments/logo"/>
  </div>

  <div id="container" class="container">
    <div layout:fragment="content" id="content">
       
    </div>
    <div th:replace="newschool/fragments/footer"/>
  </div>

  <div th:insert="newschool/fragments/bottom"/>
</body>
```

There are two principal differences between the default layout and ours:

1. The paths to the fragments will be changed from `fragments/[fragment-name]` to `newschool/fragments/[fragment-name]`. By default, Thymeleaf interprets all relative file paths as if they were rooted at the `templates` directory. If we do not make this change, we will be including the fragments from the default layout (rooted at `templates`), not our custom layout (rooted at `templates/newschool`).
2. We will move our logo from the responsive, fixed-width container that the other content resides in into its own, fluid-width container. This is a Bootstrap-specific change that will result in the logo always being as wide

as the entire browser viewport, regardless of how wide the other page content is.

Updating the fragments

In addition to updating the content part of the page in the layout template, we need to make some changes to the re-usable fragments that it includes.

The logo fragment

The default logo fragment (`fragments/logo.html`) provides an Apereo logo:

```
<header>
  <a id="logo" href="http://www.apereo.org" th:title="#{logo.titl
e}">Apereo</a>
  <h1>Apereo Central Authentication Service (CAS)</h1>
</header>
```

Our custom logo fragment (`newschool/fragments/logo.html`) will replace this with the New School SVG logo definition from the mock-up login page:

```
<!-- New School header logo lockup -->
<header role="banner" class="tns-header">
  <section class="tns-header-region">
    <div class="tns-lockup">
      <div class="tns-banner text-center">
        <h1 class="tns-uname">
          <!-- Generator: Adobe Illustrator 18.1.0, SVG Export Plug-I
n.
               SVG Version: 6.00 Build 0) -->
          <svg xmlns="http://www.w3.org/2000/svg" xml:space="preserv
e"
               version="1.1" x="0px" y="0px" viewBox="0 0 224.4 30.2"
               enable-background="new 0 0 224.4 30.2"
               id="tns-logo-svg">
            <g id="Layer_1">
              <g>
                <rect x="4.4" y="19.7" width="220" height="3.5"/>
                <rect x="4.4" y="26.7" width="220" height="3.5"/>
                <g>
                  <path d="M9,3.9v12.4H5V3.9H0V0.3h14v3.6H9z"/>
                  <path d="M29.7,10.2H19.2v6.1h-4v-16h4v6.1h10.5V0.3h
4v16h-4V10.2z"/>
                  <path d="M35.5,16.3v-16h11.4v3.6h-7.4v2.7h6V10h-6v
2.7h7.4v3.6H35.5z"/>
                  <path d="M57.7,6.1h-0.6v10.2h-4v-16h4.7l9.3,10h0.6
v-10h4v16h-4.5L57.7,6.1z"/>
                  <path d="M73.5,16.3v-16h11.4v3.6h-7.4v2.7h6V10h-6v
2.7h7.4v3.6H73.5z"/>
                  <path d="M85.2,0.3h4.5l3.5,12.7h0.9l3.2-12.7h7.5l
3.6,12.7h0.9l3-12.7h4.5l-4.4,16h-7.1l-3.7-12.7h-1.1l-3.4,12.7h-7.1L8
5.2,0.3z"/>
                  <path d="M121.5,5.2c0-3.4,2.7-5.2,6.2-5.2c
2,0,4.2,0.5,5.5,1.2l-0.8,3.4c-1.3-0.8-3.3-1.2-4.9-1.2
c-1.2,0-2.1,0.5-2.1,1.3c0,2.6,8.2,1.1,8.2,6.5c0,2.9-2,5.4-6.3,5.4
c-1.8,0-3.9-0.2-5.5-1L122,12c1.7,0.8,3.5,1.3,5.5,1.3c
1.8,0,2.2-0.6,2.2-1.4C129.7,9.5,121.5,10.9,121.5,5.2z"/>
                  <path d="M146.9,16.1c-1.3,0.5-2.4,0.5-4.2,0.5
c-5,0-8.3-3.9-8.3-8.3c0-4.5,3.5-8.3,8.7-8.3c1.3,0,2.8,0,4.1,0.5
l-0.4,3.6c-1.4-0.5-2.4-0.5-3.6-0.5c-3,0-4.7,1.7-4.7,4.7c
0,3,2,4.7,4.8,4.7c1.8,0,2.4-0.2,3.6-0.6V16.1z"/>
                  <path d="M163.2,10.2h-10.5v6.1h-4v-16h4v6.1h10.5V
0.3h4v16h-4V10.2z"/>
                  <path d="M177.8,0c5.5,0,9.5,2.8,9.5,8.2c
0,5.6-4.3,8.4-9.5,8.4c-5.2,0-9.5-2.8-9.5-8.4C168.2,3,172.1,0,177.8,0
z M177.7,13.3c3.1,0,5.4-1.6,5.4-5.1c0-3.2-2.3-4.8-5.4-4.8
s-5.4,1.6-5.4,4.9C172.4,11.7,174.6,13.3,177.7,13.3z"/>
                  <path d="M197.5,0c5.5,0,9.5,2.8,9.5,8.2c
```

```
0,5.6-4.3,8.4-9.5,8.4c-5.2,0-9.5-2.8-9.5-8.4C187.9,3,191.8,0,197.5,0
z M197.4,13.3c3.1,0,5.4-1.6,5.4-5.1c0-3.2-2.3-4.8-5.4-4.8
s-5.4,1.6-5.4,4.9C192.1,11.7,194.3,13.3,197.4,13.3z"/>
                <path d="M208,0.3h4v12.2h12.4v3.8H208V0.3z"/>
              </g>
            </g>
          </g>
        </svg>
      </h1> <!-- tns-uname -->
    </div> <!-- tns-banner -->
  </div> <!-- tns-lockup -->
 </section> <!-- tns-header-region -->
</header> <!-- tns-header -->
```

The footer fragment

The default footer fragment (`fragments/footer.html`) provides a copyright message and a “powered by” line that includes the CAS server version information (note the use of Thymeleaf text modifiers and value substitutions):

```
<footer>
  <div id="copyright" class="container">
    <p th:utext="#{copyright}"></p>
    <p>Powered by <a href="http://www.apereo.org/cas">
      Apereo Central Authentication Service
      <span th:text="${T(org.apereo.cas.util.CasVersion).getVersio
n()}"></span>
      <span th:text="${T(org.apereo.cas.util.CasVersion).getDateTim
e()}"></span> </a>
    </p>
  </div>
</footer>
```

We don’t want to provide detailed information about the server on the login page, and arguably, we shouldn’t be providing a copyright message there either since we’ll be fronting third-party services that aren’t our intellectual property, so… we can just dispense with the footer all together by commenting it out in our custom fragment (`newschool/fragments/footer.html`):

```
<!--
<footer>
  <div id="copyright" class="container">
    <p th:utext="#{copyright}"></p>
    <p>Powered by <a href="http://www.apereo.org/cas">
      Apereo Central Authentication Service
      <span th:text="${T(org.apereo.cas.util.CasVersion).getVersio
n()}"></span>
      <span th:text="${T(org.apereo.cas.util.CasVersion).getDateTim
e()}"></span> </a>
    </p>
  </div>
</footer>
-->
```

The bottom fragment

The bottom fragment, as envisioned by the CAS developers, is used to include JavaScript code that should not be executed until the entire page has loaded. They use this to check that several of the packages they need are loaded, and use a package called HeadJS to load them if they're not:

```
<script th:src="@{#{webjars.headmin.js}}"></script>
<script type="text/javascript" th:src="@{${#themes.code('cas.javascri
pt.file')}}"></script>

<script th:inline="javascript">
head.ready(document, function () {
    if (!window.jQuery) {
        var jqueryUrl = /*[[@{#{webjars.jquerymin.js}}]]*/;
        head.load(jqueryUrl, loadjQueryUI);
    } else {
        notifyResourcesAreLoaded(resourceLoadedSuccessfully);
    }
});

function loadjQueryUI() {
        var jqueryUrl = /*[[@{#{webjars.jqueryui.js}}]]*/;
        head.load(jqueryUrl, loadjQueryCookies);
}

function loadjQueryCookies() {
        var jqueryUrl = /*[[@{#{webjars.jquerycookie.js}}]]*/;
        head.load(jqueryUrl, notifyResourcesAreLoaded(resourceLoadedS
uccessfully));
}

function notifyResourcesAreLoaded(callback) {
    if (typeof callback === "function") {
        callback();
    }
}
</script>
```

For our custom bottom fragment, we will add two things:

1. We specifically include the default theme's `cas.js`, so that we don't have to duplicate those functions in our own `newschool.js` file. This approach allows us to benefit from any fixes/updates the CAS developers make to these functions; if we copied them into `newschool.js` we would not receive those updates (unless we manually checked for and applied them) .
2. We call the *Material Design for Bootstrap* library's `init()` function.

```
<script th:src="@{#{webjars.headmin.js}}"></script>
<script th:src="@{/js/cas.js}"></script>
<script type="text/javascript" th:src="@{${#themes.code('cas.javascri
pt.file')}}"></script>

<script th:inline="javascript">
  head.ready(document, function () {
    if (!window.jQuery) {
      var jqueryUrl = /*[[@{#{webjars.jquerymin.js}}]]*/;
      head.load(jqueryUrl, loadjQueryUI);
    } else {
      notifyResourcesAreLoaded(resourceLoadedSuccessfully);
    }
  });

  function loadjQueryUI() {
    var jqueryUrl = /*[[@{#{webjars.jqueryui.js}}]]*/;
    head.load(jqueryUrl, loadjQueryCookies);
  }

  function loadjQueryCookies() {
    var jqueryUrl = /*[[@{#{webjars.jquerycookie.js}}]]*/;
    head.load(jqueryUrl, notifyResourcesAreLoaded(resourceLoadedSucce
ssfully));
  }

  function notifyResourcesAreLoaded(callback) {
    if (typeof callback === "function") {
      callback();
    }
  }
</script>

<script>
  $(function () {
    $.material.init();
  });
</script>
```

Update the login view

The CAS login view is built from a content template (`casLoginView.html`), a fragment (`fragments/loginform.html`), and some message strings defined in `messages.properties` (or `messages_xx.properties` or `custom_messages.properties`). When the content template is substituted into the Thymeleaf layout template (`layout.html`), the result will be the CAS login page that is displayed to the user.

Updating the login view content template

The default login view (`templates/casLoginView.html`) defines two columns on the page: the left-hand column contains the actual login dialog box, and the right-hand column contains a number of “additional information” boxes. All of these elements are defined by different fragments.

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" layou
t:decorate="~{layout}">

<head>
  <title th:text="#{cas.login.pagetitle}"></title>
</head>

<body id="cas" class="login">
  <div layout:fragment="content">
    <div class="row">
      <div id="notices" class="col-sm-12 col-md-6 col-md-push-6">
        <div th:replace="fragments/insecure"/>
        <div th:replace="fragments/defaultauthn"/>
        <div th:replace="fragments/cookies"/>
        <div th:replace="fragments/serviceui"/>
        <div th:replace="fragments/cas-resources-list" />
        <div th:replace="fragments/loginProviders" />
      </div>
      <div class="col-sm-12 col-md-6 col-md-pull-6">
        <div th:replace="fragments/loginform" />
      </div>
    </div>
  </div>
</body>
</html>
```

As a reminder, this looks something like this (not all of the fragments in the right-hand column are displayed by default):

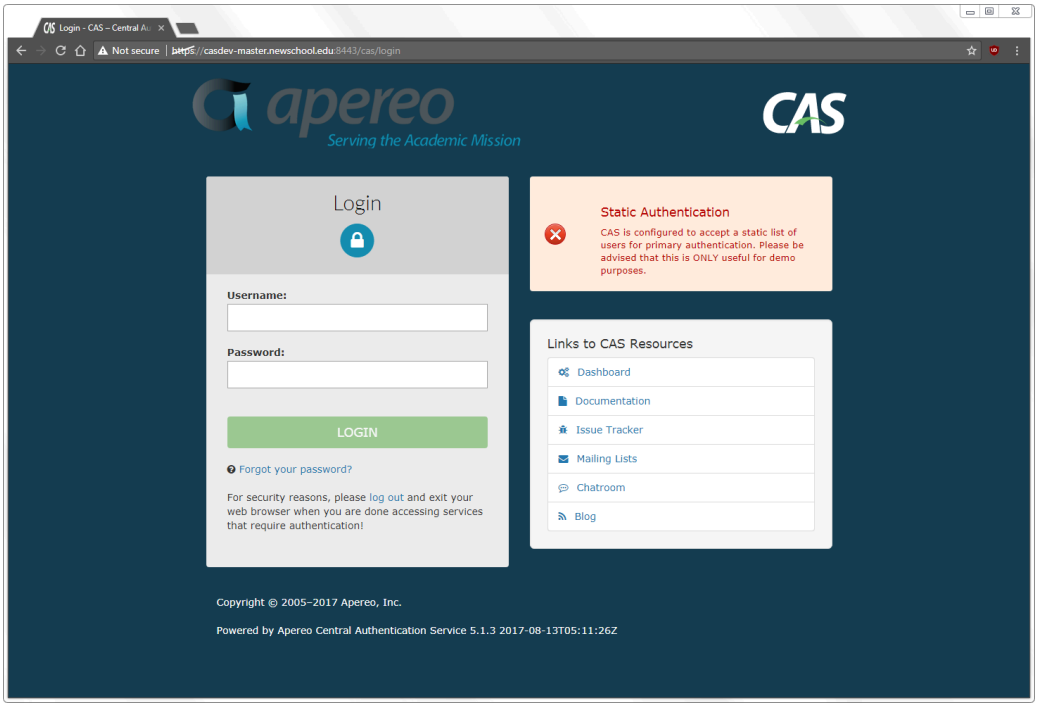

Figure 26. The default CAS server login page

Our login view (`templates/newschool/casLoginView.html`) will be a little simpler, with just one fragment, the login dialog box:

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" layou
t:decorate="~{newschool/layout}">

<head>
  <title th:text="#{cas.login.pagetitle}"></title>
</head>

<body id="cas" class="login">
  <div layout:fragment="content">
    <div class="row">
      <div class="col-sm-12 col-md-4 col-md-offset-4">
        <div th:replace="newschool/fragments/loginform"/>
      </div>
    </div>
  </div>
</body>
</html>
```

As explained previously, Thymeleaf interprets all relative file paths as if they were rooted at the `templates` directory. Therefore, aside from removing the extra fragments that we're not going to use, we have to make two other changes in our template:

1. In the `layout:decorate` attribute of the `<html>` tag, we will replace `~{layout}` with `~{newschool/layout}`. If we do not make this change, our content template will be applied to the default layout template (`templates/layout.html`) rather than our custom layout (`templates/newschool/layout.html`).
2. In the `th:replace` attribute of the inner-most `<div>` tag, we will replace `fragments/loginform` with `newschool/fragments/loginform`. If we did not make this change, the default login form fragment would be included instead of our customized version.

Updating the login form fragment

To create our login form fragment (`templates/newschool/fragments/loginform.html`), we will start with the "login form box" section of the mock-up page. From there, we will replace "hard coded" text strings with references to message property names, and incorporate any "interesting" or "desirable" features from the default login form fragment or one of the other fragments that our login view is not including. The result will be something like this:

```
<div class="well" id="login">
  <header class="tns-header">
    <div class="tns-banner">
      <h2 class="tns-sitename">Single Sign-On</h2>
    </div>
  </header>

  <h1 th:text="#{cas.login.pagetitle}"></h1>

  <div class="registered-service" th:if="${registeredService}">
    <div th:if="${serviceUIMetadata}">
      <p>to continue to
         <span th:text="${serviceUIMetadata.displayName}"></span></p>
    </div>
    <div th:unless="${serviceUIMetadata}">
      <p>to continue to
        <span th:text="${registeredService.name}"></span></p>
    </div>
  </div> <!-- registered-service -->

  <form method="post" id="fm1" th:object="${credential}" action="logi
n">
    <div class="alert-row">
      <div class="alert alert-danger" th:if="${#fields.hasError
s('*')}">
        <span th:each="err : ${#fields.errors('*')}" th:utext="${er
r}"/>
      </div>
    </div> <!-- alert-row -->

    <div class="form-group label-floating is-empty">
      <label class="control-label" for="username"
        th:utext="#{screen.welcome.label.netid}"/>
      <div th:if="${openIdLocalId}">
        <strong>
          <span th:utext="${openIdLocalId}"/>
        </strong>
        <input type="hidden"
               id="username"
               name="username"
               th:value="${openIdLocalId}"/>
      </div>
      <div th:unless="${openIdLocalId}">
        <input class="required form-control"
               id="username"
               name="username"
               size="25"
```

```
                tabindex="1"
                type="text"
                th:disabled="${guaEnabled}"
                th:field="*{username}"
                th:accesskey="#{screen.welcome.label.netid.accesskey}"
                autocomplete="off"/>
      </div>
    </div> <!-- form-group -->

    <div class="form-group label-floating is-empty" style="padding-bo
ttom: 0px">
      <label class="control-label" for="password"
        th:utext="#{screen.welcome.label.password}"/>
      <div>
        <input class="required form-control"
                type="password"
                id="password"
                name="password"
                size="25"
                tabindex="2"
                th:accesskey="#{screen.welcome.label.password.accesske
y}"
                th:field="*{password}"
                autocomplete="off"/>
        <div class="capslock-msg">
          <p id="capslock-on" style="display: none">
            <i class="material-icons">error</i>
            <span th:utext="#{screen.capslock.on}"/>
          </p>
        </div>
      </div>
    </div> <!-- form-group -->

    <div class="form-row continue-button" th:if="${recaptchaSiteKe
y}">
      <div class="g-recaptcha" th:attr="data-sitekey=${recaptchaSiteK
ey}"/>
    </div> <!-- form-row -->
    <div class="form-row text-center">
      <input type="hidden" name="execution" th:value="${flowExecution
Key}"/>
      <input type="hidden" name="_eventId" value="submit"/>
      <input type="hidden" name="geolocation"/>

      <input class="btn btn-primary btn-lg" style="margin-top: 0px"
             name="submit"
             accesskey="l"
```

```
            th:value="#{screen.welcome.button.login}"
            tabindex="6"
            type="submit"/>
      <div class="ripple-container"></div>
    </div> <!-- form-row -->

    <div class="form-row account-options">
      <p>
        <a th:href="#{screen.welcome.resetPassword.url}">
          <i class="material-icons">lock</i>
          <span th:utext="#{screen.welcome.resetPassword.text}"/>
        </a>
      </p>
      <p>
        <a th:href="#{screen.welcome.lookupNetID.url}">
          <i class="material-icons">help</i>
          <span th:utext="#{screen.welcome.lookupNetID.text}"/>
        </a>
      </p>
    </div> <!-- form-row -->

    <div class="form-row text-right privacy-terms">
      <a th:href="#{screen.welcome.privacy.url}" target="_blank">
        <span th:utext="#{screen.welcome.privacy.text}"/>
      </a>
      <a th:href="#{screen.welcome.terms.url}" target="_blank">
        <span th:utext="#{screen.welcome.terms.text}"/>
      </a>
    </div> <!-- form-row -->
  </form>
</div>
```

Some of the changes that will be made from the mock-up include:

- **Line 7 (`<h1>` tag).** Replace the hard coded “Log in” with a reference to a message property.
- **Lines 10-19 (`registered-service` class).** Incorporate some Thymeleaf code from the default `serviceui` fragment that obtains the name of the service (as defined in the service registry) the user is attempting to access. The `if` / `unless` sequence tests to see if the service is a SAML2-based service or a CAS-based service and obtains the name from the appropriate variable.
- **Lines 22-26 (`alert-row` class).** Incorporate some Thymeleaf code from the default `loginform` fragment that displays any form input error

messages.

- **Lines 28-74 (username and password prompts).** Include the more complex username and password prompt code from the default `loginform` fragment. This includes support for OpenID, caps lock detection, etc.
- **Lines 93-115 (links for password reset, NetID lookup, privacy, and terms).** Replace the hard coded text for these links, as well as the links themselves, with message property references.

Updating the text strings

As noted above, the login form fragment uses message properties for all of its text, rather than hard coded values. Some of these properties are defined by the default CAS message bundles (`messages.properties` for the `en` locale and `messages_xx.properties` for other locales). We need to change the values of some of these properties to better match our environment (for example, we prefer the term “NetID” to “Username”). Other properties are specific to our login form and are not included in the default message bundles; we will have to define those properties ourselves. We can use the same file to accomplish both of these tasks; properties defined in the `WEB-INF/classes/custom_messages.properties` file will override properties of the same name defined in `messages.properties` or `messages_xx.properties`. So, our `custom_messages.properties` file will look something like this:

```
# Just in case (we don't display this currently)
copyright=Copyright &copy; 2018 The New School

# Page title
cas.login.pagetitle=Log in

# Replace "Username" with "NetID"
screen.welcome.label.netid=<span class="accesskey">N</span>etID:
screen.welcome.label.netid.accesskey=n

# Replace "LOGIN" with "Continue"
screen.welcome.button.login=Continue

# Text and url for password reset
screen.welcome.resetPassword.url=https://account.newschool.edu/cgi-bi
n/acctservices.pl?f=i&s=pr
screen.welcome.resetPassword.text=Reset your password

# Text and url for NetID lookup
screen.welcome.lookupNetID.url=https://account.newschool.edu/cgi-bin/
acctservices.pl?f=i&s=gn
screen.welcome.lookupNetID.text=Look up your NetID

# Text and url for the "privacy" link
screen.welcome.privacy.url=//www.newschool.edu/privacy-policy/
screen.welcome.privacy.text=Privacy

# Text and url for the "terms" link
screen.welcome.terms.url=//it.newschool.edu/sites/default/files/uploa
ds/documents/Statement%20on%20the%20Responsibilities%20of%20Compute
r%20Users%20v2.1.pdf
screen.welcome.terms.text=Terms

# Text to display when caps lock is on
screen.capslock.on=Caps Lock is on
```

Update the logout view

The logout view is displayed when a user logs out of the CAS service, i.e., when his or her browser is directed to the `/cas/logout` endpoint. Unlike the default login view, the default logout view (`templates/casLogoutView.html`) does not include any fragments, it just displays some message text:

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" layou
t:decorate="~{layout}">

<head>
  <title th:text="#{screen.logout.header}"></title>
</head>

<body id="cas">
<div layout:fragment="content">
  <div class="alert alert-success">
    <h2 th:utext="#{screen.logout.header}"/>
    <p th:utext="#{screen.logout.success}"/>
    <p th:utext="#{screen.logout.security}" />
  </div>
</div>
</body>
</html>
```

Updating the logout view template

For our custom logout view (`templates/newschool/casLogoutView.html`), the only change we really need to make is to replace the default layout template with our custom template:

```
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" layou
t:decorate="~{newschool/layout}">
```

Although not strictly necessary, we will also change a couple of the text strings to better match our local environment. We can do this by overriding their values in `WEB-INF/classes/custom_messages.properties` :

```
screen.logout.success=You have successfully logged out of the New Sch
ool \
  Single Sign-On Service. You may <a href="login">log in</a> again.
screen.logout.security=For security reasons, please exit your web bro
wser.
```

Update other relevant views

Once the layout template and the login view have been customized, most of the difficult user interface customization work is done. However, there are a number of other views that should be customized to maintain a consistent look and feel. At a minimum, these views should be modified to use the custom Thymeleaf layout template instead of the default:

```
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" layou
t:decorate="~{newschool/layout}">
```

Depending on the particular view, it may also be desirable to update some of the text displayed in the view by redefining the relevant properties in `WEB-INF/classes/custom_messages.properties`.

Views that may need to be customized

Some of the likely candidates in `templates/newschool` for customization include:

- **Generic views.** CAS provides a “generic success” view (`casGenericSuccessView.html`) that is displayed when a user logs directly into the CAS server without having been directed there by another service (much like we did when first building the server). It also provides a “service error” view (`casServiceErrorView.html`) that is displayed when there is an error with the service (usually an attempt to use a service that is not in the service registry).
- **MFA views.** CAS provides one or more views for each multi-factor authentication product that it supports: `casDuoLoginView.html`, `casGoogleAuthenticatorLoginView.html`, `casGoogleAuthenticatorRegistrationView.html`, etc. Since these become part of the login web flow, they should have the same look and feel as the “base” login page.
- **LPEE views.** CAS provides views for the various LDAP Password Policy Enforcement (LPPE) outcomes: `casAccountDisabledView.html`, `casAccountLockedView.html`, `casExpiredPassView.html`, etc. These become part of the login web flow if LPPE support is enabled.
- **Password management views.** CAS provides several views associated with its (optional) user password management features: `casMustChangePassView.html`, `casPasswordUpdateSuccessView.html`, `casResetPasswordSendInstructionsView.html`, etc. These may also

become part of the login web flow.

There are a variety of other views included with CAS that are not mentioned above; generally, any feature that will result in interaction with a user will have a view (or multiple views) associated with it. The files containing the views are, for the most part, named in a manner that should make it obvious which feature they belong to. It's only necessary to update those views that belong to enabled features.

Error views

The `templates/newschool/error.html` view is used to display a variety of error messages that may occur during interaction with the user; this view should be customized as appropriate.

The views in the `templates/newschool/error` subdirectory (`401.html` , `403.html` , `404.html` , `405.html` , and `423.html`) are displayed by Tomcat when the associated HTTP error occurs; these should be customized with at least the custom layout template.

Dashboard views

The `templates/newschool/monitoring` subdirectory contains the views associated with the CAS dashboard (admin pages). These pages come with their own layout template (`templates/newschool/monitoring/layout.html`); they do not use the same template as the "user" views. It's not necessary to customize these views, but it may be desirable depending on the audience that will be using them.

Install and test the final result

Once the custom theme files have been finalized (or at least reached a steady intermediate state), they can be copied back into the overlay and the CAS server can be rebuilt and deployed for more extensive testing.

Copy the “live” files back into the overlay

To begin, the “live” copy of the custom theme files must be copied back into the overlay. Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay
casdev-master# cd src/resources
casdev-master# ssh casdev-srv01 "cd /var/lib/tomcat/cas/WEB-INF/class
es; tar cf - newschool.properties custom_messages.properties static/t
hemes/newschool templates/newschool" | tar xf -
casdev-master# chown -R root.root .
casdev-master# chmod -R og+rX .
```

on the master build server (***casdev-master***) to copy the “live” files from the server where they were being edited (***casdev-srv01*** in the example) back into the overlay.

Rebuild the server

Run Maven to rebuild the server to include the new theme and view files:

```
casdev-master# cd ../..
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 01:00 min
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 35M/84M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

Install and test on the master build server

Use the scripts created earlier (page 99) (or repeat the commands) to install the updated CAS configuration files on the master build server (***casdev-master***):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the scripts created earlier (page 99):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Shut down all but one of the pool servers

Operating CAS with a pool of servers instead of a single server requires special configuration. Because that configuration hasn't been completed yet, testing must be performed against a single server. Further, since we're going to be editing the CSS, JavaScript, and HTML files and looking for those changes, we want to make sure we're accessing the same server (the one where we're editing the files) every time. Therefore, the other servers in the pool should be shut down so that the load balancer will direct all traffic to that single server. Run the command

```
# systemctl stop tomcat
```

on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

Commit changes to Git

Before moving on to the next stage of development, commit the new custom user interface files to Git to make changes easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add src
casdev-master# git commit -m "Created New School-branded user interfa
ce"
[master (root-commit) e2cb175] Created New School-branded user interf
ace
 137 files changed, 5243 insertions(+)
 create mode 100644 src/main/resources/custom_messages.properties
 create mode 100644 src/main/resources/newschool.properties
 create mode 100644 src/main/resources/static/themes/newschool/css/ad
min.css
 (lots of output...)
 create mode 100644 src/main/resources/templates/newschool/protocol/o
penid/casOpenIdServiceFailureView.html
 create mode 100644 src/main/resources/templates/newschool/protocol/o
penid/casOpenIdServiceSuccessView.html
 create mode 100644 src/main/resources/templates/newschool/protocol/o
penid/user.html
casdev-master#
```

on the master build server (***casdev-master***).

High availability

Summary: Now that everything is working correctly in a single-server environment, the steps to enable multiple servers to operate in a pooled configuration can be performed.

As explained in the introduction (page 9), one of the implementation goals for this environment is to have “high availability (fault-tolerant) everything.” To that end, the environment has been built with a pool of servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) behind a load balancer. To enable these servers to work together in an active-active configuration, where any server in the pool is capable of servicing any request and the pool can continue to service requests even if one or more servers is unavailable, the following tasks must be performed:

1. A distributed ticket registry (cache) that replicates all tickets to all servers must be created to ensure that a ticket can be located from any server (the server that is asked to validate a ticket may not be the same server that originally created it).
2. A distributed service registry that replicates all registered services to all servers must be created to ensure that all servers support the same set of services.
3. Distributed storage for CAS SAML IdP metadata must be created to ensure that all the servers behave the same way, and distributed storage for cached SAML SP metadata must be created to ensure that all servers know about all SPs.
4. A distributed configuration property storage solution that replicates all configuration settings to all servers must be created to ensure that all servers are configured the same way.

CAS 5 supports a variety of caches, databases, and configuration servers to implement ticket registries, service registries, and configuration property storage. For our implementation, we will use the MongoDB NoSQL database, which offers a lightweight implementation with built-in replication and fault tolerance features that can be installed on the same virtual machines that we’re using to run the CAS servers.

References

- CAS 5: Ticketing
- CAS 5: Service Management

- CAS 5: Configuration Server

Install and configure MongoDB

Summary: MongoDB will be used to store the ticket registry, service registry, and configuration properties for all CAS servers in the environment.

MongoDB is an open-source NoSQL database. MongoDB stores data records as JSON-like *documents* that contain field-value pairs. The value of a field can be any of several different data types such as numbers, strings, booleans, dates, objects, and arrays. In MongoDB, *databases* hold *collections* of documents. Collections are somewhat analogous to tables in relational databases, but a collection does not require its documents to have the same schema; i.e. the documents in a single collection do not all have to have the same set of fields and the data type for a field can differ from one document to another within a collection.

MongoDB is a distributed database by design, so high availability, horizontal scaling, and geographic distribution are built in and easy to use. A *replica set* is a group of MongoDB instances that manage the same data set (group of databases). Replica sets provide redundancy and high availability, and are the basis for all production MongoDB deployments. A replica set contains multiple data storage nodes and, optionally, an arbiter node (used when needed to ensure there are an odd number of members in the replica set).

One and only one of the data storage nodes is deemed the primary node, and the others are deemed secondary nodes (or arbiters). The primary node receives all write operations (and usually, all read operations as well). The primary records all changes to its data set in a transaction log. The secondary nodes copy and apply these changes in an asynchronous process, resulting in the same data set being stored on multiple servers. If the primary server is unavailable, the secondaries will hold an election to elect one of themselves as the new primary.

In this section, we will install the latest, stable version of MongoDB on each of the CAS servers in the environment (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***), and then group those servers together as a replica set. We will also implement security controls to prevent access to the servers (and their data) from unauthorized sources, since some of the data stored in the database may be sensitive (e.g., passwords in configuration properties).

Install the MongoDB software

Red Hat does not offer an up-to-date version of MongoDB on RHEL 7, but MongoDB, Inc. offers its own set of repositories from which the most current version can be installed.

> ℹ **Note:** Although the master build server will not be a member of the replica set, the MongoDB software will be installed there to facilitate customizing configuration files and distributing them to the replica set members, and to enable use of the `mongo` shell client application. In each section below, instructions are provided to indicate on which server(s) that step should be performed.

Install the MongoDB repository

As of this writing the latest, stable version of MongoDB is 3.6, originally released in December 2017. To install MongoDB 3.6 with `yum`, create the file `/etc/yum.repos.d/mongodb-org-3.6.repo` on the master build server (***casdev-master***) with the following contents:

```
[mongodb-org-3.6]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/3.6/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.6.asc
```

Then copy the new file to each of the CAS servers by running the commands

```
casdev-master# for i in 01 02 03
> do
> scp -p /etc/yum.repos.d/mongodb-org-3.6.repo casdev-srv${i}:/etc/yu
m.repos.d/mongodb-org-3.6.repo
> done
mongodb-org-3.6.repo                          100%  200   273.9KB/
s   00:00
mongodb-org-3.6.repo                          100%  200   251.1KB/
s   00:00
mongodb-org-3.6.repo                          100%  200   255.4KB/
s   00:00
casdev-master#
```

Install MongoDB

Run the command

```
# yum -y install mongodb-org
```

on the master build server (***casdev-master***) and each of the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) to install MongoDB.

Correct directory permissions

The default MongoDB installation creates the MongoDB data directory and MongoDB log directory with permissions that allow all users on the system to access them. However, only the `mongod` user and `mongod` group actually need access to these directories, so access for other users should be removed. Run the command

```
# chmod -R o= /var/lib/mongo /var/log/mongodb
```

on the master build server (***casdev-master***) and each of the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) to correct the directory permissions.

Configure `logrotate`

By default, `mongod` will keep writing to the same log file until it is told not to; there is no pre-configured scheme for rotating logs. To correct this, configure the `logroate` program, which is probably already being used to rotate various system log files, to rotate `/var/log/mongodb/mongod.log` as well.

Create the file `/etc/logrotate.d/mongod` on the master build server (***casdev-master***) with the following contents:

```
/var/log/mongodb/mongod.log
{
    daily
    dateext
    dateformat -%Y-%m-%d
    dateyesterday
    extension .log
    missingok
    notifempty
    rotate 30
    create 0640 mongod mongod
    sharedscripts
    postrotate
        /bin/kill -SIGUSR1 $(cat /var/run/mongodb/mongod.pid)
    endscript
}
```

This will rotate `mongod.log` every day, moving the current log file to `mongod-YYYY-MM-DD.log`. Since `logrotate` typically runs in the wee hours of the morning, the `dateyesterday` directive tells it to use yesterday's date for the file name, since that's when most of the log entries will come from. The last 30 days' worth of log files will be kept; older files will be deleted automatically. Once the file has been rotated, `logrotate` will signal `mongod` to switch to the new log file.

Run the command

```
casdev-master# logrotate -df /etc/logrotate.d/mongod
reading config file /etc/logrotate.d/mongod
extension is now .log
Allocating hash table for state file, size 15360 B

Handling 1 logs

rotating pattern: /var/log/mongodb/mongod.log
 forced from command line (30 rotations)
empty log files are not rotated, old logs are removed
considering log /var/log/mongodb/mongod.log
  log needs rotating
rotating log /var/log/mongodb/mongod.log, log->rotateCount is 30
Converted ' -%Y-%m-%d' -> '-%Y-%m-%d'
dateext suffix '-2018-04-29'
glob pattern '-[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]'
glob finding old rotated logs failed
fscreate context set to system_u:object_r:mongod_log_t:s0
renaming /var/log/mongodb/mongod.log to /var/log/mongodb/mongod-201
8-04-29.log
creating new /var/log/mongodb/mongod.log mode = 0640 uid = 994 gid =
992
running postrotate script
running script with arg /var/log/mongodb/mongod.log
: "
        /bin/kill -SIGUSR1 $(cat /var/run/mongodb/mongod.pid)
"
casdev-master#
```

to check the syntax of the file and confirm that it will do what you expect. Then copy the new file to each of the CAS servers by running the commands

```
casdev-master# for i in 01 02 03
> do
> scp -p /etc/logrotate.d/mongod casdev-srv${i}:/etc/logrotate.d/mong
od
> done
mongod                                        100%  293   400.8KB/
s   00:00
mongod                                        100%  293   363.6KB/
s   00:00
mongod                                        100%  293   397.7KB/
s   00:00
casdev-master#
```

Disable the `mongod` service on the master build server

Since the master build server will not be part of the replica set, it does not need to run the MongoDB server (`mongod`). Run the command

```
casdev-master# systemctl disable mongod
Removed symlink /etc/systemd/system/multi-user.target.wants/mongod.se
rvice.
casdev-master#
```

on the master build server (***casdev-master***) to prevent `mongod` from being started when the system boots.

References

MongoDB: Install MongoDB Community Edition on Red Hat Enterprise or CentOS Linux

Disable Transparent Huge Pages

> **Note:** This step is only necessary when running MongoDB on Linux servers (physical or virtual).

Transparent Huge Pages (THP) is a Linux memory management feature designed to reduce the overhead of translation lookaside buffer (page table) lookups on machines with large amounts of memory by using larger virtual memory pages. However, THP often causes performance problems for database workloads, because they tend to have sparse, rather than contiguous, memory access patterns. MongoDB, Inc. recommends disabling THP on servers running MongoDB.

Define a service unit to disable THP

Create a file on the master build server (***casdev-master***) called `/etc/systemd/system/mongod-disable-thp.service` with the following contents:

```
[Unit]
Description="Disable Transparent Huge Pages (THP) before mongod start
s"
Before=mongod.service

[Service]
Type=oneshot
ExecStart=/bin/sh -c 'echo never > /sys/kernel/mm/transparent_hugepag
e/enabled'
ExecStart=/bin/sh -c 'echo never > /sys/kernel/mm/transparent_hugepag
e/defrag'

[Install]
RequiredBy=mongod.service
```

Install and enable the service unit

Run the commands

```
casdev-master# restorecon /etc/systemd/system/mongod-disable-thp.serv
ice
casdev-master# chmod 644 /etc/systemd/system/mongod-disable-thp.servi
ce
casdev-master# for i in 01 02 03
> do
> scp -p /etc/systemd/system/mongod-disable-thp.service casdev-sr
v${i}:/etc/systemd/system/mongod-disable-thp.service
> ssh casdev-srv${i} systemctl enable mongod-disable-thp
> done
mongod-disable-thp.service                    100%  321   984.7KB/
s   00:00
Created symlink from /etc/systemd/system/mongod.service.requires/mong
od-disable-thp.service to /etc/systemd/system/mongod-disable-thp.serv
ice.
mongod-disable-thp.service                    100%  321     1.0MB/
s   00:00
Created symlink from /etc/systemd/system/mongod.service.requires/mong
od-disable-thp.service to /etc/systemd/system/mongod-disable-thp.serv
ice.
mongod-disable-thp.service                    100%  321   441.2KB/
s   00:00
Created symlink from /etc/systemd/system/mongod.service.requires/mong
od-disable-thp.service to /etc/systemd/system/mongod-disable-thp.serv
ice.
casdev-master#
```

to copy the service unit to the CAS servers and enable it to be executed by `systemd`. Do not enable the service unit on the master build server (***casdev-master***), since `mongod` will not be running there.

References

MongoDB: Disable Transparent Huge Pages (THP)

Open MongoDB port in the firewall

To enable the `mongod` processes in the replica set to communicate with each other, the MongoDB port (TCP 27017) must be opened in the firewall on each of the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***).

Create a `firewalld` service configuration

First, create a `firewalld` service configuration file on the master build server (***casdev-master***) called `/etc/firewalld/services/mongod.xml` with the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>mongod</short>
  <description>MongoDB default port for mongod and mongos instance
s.</description>
  <port protocol="tcp" port="27017"/>
</service>
```

to define the service, and then run the commands

```
casdev-master# restorecon /etc/firewalld/services/mongod.xml
casdev-master# chmod 640 /etc/firewalld/services/mongod.xml
casdev-master# firewall-cmd --reload
success
casdev-master#
```

to assign the correct SELinux context and file permissions to the `mongod.xml` file and inform `firewalld` of its existence. Then copy the new file to each of the CAS servers and inform their `firewalld` processes of its existence by running the commands

```
casdev-master# for i in 01 02 03
> do
> scp -p /etc/firewalld/services/mongod.xml casdev-srv${i}:/etc/firew
alld/services/mongod.xml
> ssh casdev-srv${i} firewall-cmd --reload
> done
mongod.xml                                    100%  205   309.3KB/
s   00:00
success
mongod.xml                                    100%  205   320.6KB/
s   00:00
success
mongod.xml                                    100%  205   333.8KB/
s   00:00
success
casdev-master#
```

Configure the firewall

Because some of the information stored in MongoDB may be sensitive (e.g., passwords in configuration properties), we will only open the MongoDB port in the firewall to connections from the CAS servers and the master build server.

Create an ipset of source addresses

A `firewalld` *ipset* is a named list of IP addresses that can be referenced in firewall rules. We will define an ipset called `cas-servers` that contains the addresses of the master build server and the CAS servers, which can be used to create the firewall rule in the next section. Run the commands

```
# firewall-cmd  --permanent --new-ipset=cas-servers --type=hash:net
success
# firewall-cmd --reload
success
# firewall-cmd --permanent --ipset=cas-servers --add-entry=192.168.10
0.100
success
# firewall-cmd --permanent --ipset=cas-servers --add-entry=192.168.10
0.101
success
# firewall-cmd --permanent --ipset=cas-servers --add-entry=192.168.10
0.102
success
# firewall-cmd --permanent --ipset=cas-servers --add-entry=192.168.10
0.103
success
# firewall-cmd --reload
success
#
```

on each of the three CAS servers (***casdev-srv01***, ***casdev-srv02***. and ***casdev-srv03***). It is not necessary to define the ipset on the master build server (***casdev-master***), since it will not be running `mongod`.

Create a rich rule to enable access

In addition to command-line arguments that allow the creation of basic allow/deny rules, `firewalld` supports a *rich rule* language for creating more complex rules. The rich language extends the basic set of elements (service, port, etc.) with additional elements, such as source and destination addresses, logging, actions and limits for logs and actions. We will use a rich rule to limit connections to the `mongod` port to the IP addresses in the `cas-servers` ipset defined above. Run the commands

```
# firewall-cmd --permanent --zone=public --add-rich-rule='rule famil
y="ipv4" source ipset="cas-servers" service name="mongod" accept'
success
# firewall-cmd --reload
success
#
```

on each of the three CAS servers (***casdev-srv01***, ***casdev-srv02***. and ***casdev-srv03***). It is not necessary to install the rule on the master build server (***casdev-master***), since it will not be running `mongod`.

References

- Firewalld: IP Sets
- Firewalld: Rich Rule Language

Set up MongoDB authentication

MongoDB provides an internal authentication feature that, when enabled, will require the individual members of the replica set to authenticate to each other. MongoDB also provides role-based access control, which requires client applications (and users) to authenticate to the database with a username and password, and then sets limits on the database(s) each user may access, and the operations the user may perform there. Both of these features will be used to protect the data stored in the CAS MongoDB instance.

Create an administrative user

On one of the replica set members (e.g., ***casdev-srv01***), start `mongod` by running the commands

```
casdev-srv01# systemctl start mongod-disable-thp
casdev-srv01# systemctl start mongod
```

> ℹ **Note:** Make sure you are using one of the replica set members (CAS servers), not the master build server (***casdev-master***).

On the same server, run the `mongo` shell to connect to the server:

```
casdev-srv01# mongo
MongoDB shell version v3.6.0
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.0
Server has startup warnings:
YYYY-MM-DDTHH:MM:SS.sss-0000 I CONTROL  [initandlisten]
YYYY-MM-DDTHH:MM:SS.sss-0000 I CONTROL  [initandlisten] ** WARNING: A
ccess control is not enabled for the database.
YYYY-MM-DDTHH:MM:SS.sss-0000 I CONTROL  [initandlisten] **          R
ead and write access to data and configuration is unrestricted.
YYYY-MM-DDTHH:MM:SS.sss-0000 I CONTROL  [initandlisten]
>
```

Administrative users are created in the special `admin` database. Using the `mongo` shell, connect to the `admin` database and then create an administrative user called `mongoadmin` by running the commands

```
> use admin
switched to db admin
> db.createUser( { user: "mongoadmin", pwd: "changeit", roles: [ { ro
le: "root", db: "admin" } ] } )
Successfully added user: {
        "user" : "mongoadmin",
        "roles" : [
                {
                        "role" : "root",
                        "db" : "admin"
                }
        ]
}
>
```

Warning: The command above uses `changeit` as the value of the `mongoadmin` password. Obviously, something other than this should be used in a production MongoDB deployment.

Then exit the `mongo` shell:

```
> exit
bye
```

Generate a SCRAM-SHA1 keyfile

To implement internal authentication between the replica set members, MongoDB supports the Salted Challenge Response Authentication Mechanism (SCRAM-SHA-1) . To support this, a keyfile containing the shared secret (password) is created and installed on each replica set member server. Run the command

```
casdev-master# openssl rand -base64 756 > mongod-auth.key
```

on the master build server (***casdev-master***) to generate a random key (password). Although the master build server is not a member of the replica set, it makes sense to store a copy of the keyfile there for safekeeping. Then run the commands

```
casdev-master# tar cf kf.tar --owner=mongod --group=mongod --mode=40
0 mongod-auth.key
casdev-master# for i in 01 02 03
> do
> scp kf.tar casdev-srv${i}:/tmp/kf.tar
> ssh casdev-srv${i} "cd /var/lib/mongo; tar xf /tmp/kf.tar; rm /tmp/
kf.tar"
> done
kf.tar                                        100%   10KB 437.3KB/
s   00:00
kf.tar                                        100%   10KB   1.0MB/
s   00:00
kf.tar                                        100%   10KB 128.4KB/
s   00:00
casdev-master#
```

to distribute the keyfile to each of the replica set members with the correct owner, group, and permissions.

Update the MongoDB configuration file

MongoDB uses a YAML-formatted configuration file, `/etc/mongod.conf`. Edit this file on the master build server (***casdev-master***) and make the following changes:

1. In the `net` section, change the value of the `bindIp` setting from `127.0.0.1` (listen only on the loopback interface) to `0.0.0.0` (listen on all interfaces). This will enable the other members of the replica set to connect to the server.
2. Uncomment the `security` section and add a `keyFile` setting with the path to the keyfile created above (`/var/lib/mongo/mongod-auth.key`).
3. Also in the `security` section, add an `authorization` setting with the value `enabled` (this turns on role-based access control).
4. Uncomment the `replication` section and add a `replSetName` setting with the value `rs0` .
5. In the `systemLog` section add a `logRotate` setting with the value `reopen` (this is necessary for the `logrotate` configuration, created earlier, to work properly).

After making these changes, the affected sections of the configuration file should look like this:

```
net:
  port: 27017
  bindIp: 0.0.0.0

security:
  keyFile: /var/lib/mongo/mongod-auth.key
  authorization: enabled

replication:
  replSetName: rs0

systemLog:
  logRotate: reopen
```

Then run the commands

```
casdev-master# for i in 01 02 03
> do
> scp -p /etc/mongod.conf casdev-srv${i}:/etc/mongod.conf
> ssh casdev-srv${i} "systemctl start mongod-disable-thp; systemctl r
estart mongod"
> done
mongod.conf                                   100%  813    41.2KB/
s   00:00
mongod.conf                                   100%  813    53.4KB/
s   00:00
mongod.conf                                   100%  813   721.3KB/
s   00:00
casdev-master#
```

to copy the updated configuration file to each member of the replica set and (re)start the `mongod` server.

References

- MongoDB: Internal Authentication
- MongoDB: Role-Based Access Control

Create the replica set

The replica set is created by initiating replication on the replica set member server where the administrative user was created, and then adding the other replica set member servers to the set.

Connect with the `mongo` shell

On the replica set member where the `mongoadmin` user was created in the previous section (***casdev-srv01*** in our case), start the `mongo` shell again by running the command

```
casdev-srv01#  mongo -u mongoadmin -p --authenticationDatabase admin
MongoDB shell version v3.6.0
Enter password:
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.0
>
```

and entering the correct password (“`changeit`”).

Initiate the replica set

From the `mongo` shell, run the command

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default config
uration for the set",
        "me" : "casdev-srv01.newschool.edu:27017",
        "ok" : 1,
        "operationTime" : Timestamp(1512664653, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1512664653, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAA
AA="),
                        "keyId" : NumberLong(0)
                }
        }
}
>
```

Add members to the replica set

Continuing in the `mongo` shell, add the other members of the replica set:

```
> rs.add("casdev-srv02.newschool.edu")
{
        "ok" : 1,
        "operationTime" : Timestamp(1512664727, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1512664727, 1),
                "signature" : {
                        "hash" : BinData(0,"wbhbHdhI1gtR+SWQSh2XARQw9
jw="),
                        "keyId" : NumberLong("6496845223040122881")
                }
        }
}
> rs.add("casdev-srv03.newschool.edu")
{
        "ok" : 1,
        "operationTime" : Timestamp(1512664876, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1512664876, 1),
                "signature" : {
                        "hash" : BinData(0,"gIrttrr3VzqhM2iIWxX5ITwMI
hI="),
                        "keyId" : NumberLong("6496845223040122881")
                }
        }
}
>
```

Display the replica set configuration

To view the configuration of the replica set, use the `rs.conf()` command to the `mongo` shell:

```
> rs.conf()
{
        "_id" : "rs0",
        "version" : 3,
        "protocolVersion" : NumberLong(1),
        "members" : [
                {
                        "_id" : 0,
                        "host" : "casdev-srv01.newschool.edu:27017",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 1,
                        "host" : "casdev-srv02.newschool.edu:27017",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 2,
                        "host" : "casdev-srv03.newschool.edu:27017",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                }
        ],
        "settings" : {
```

```
                "chainingAllowed" : true,
                "heartbeatIntervalMillis" : 2000,
                "heartbeatTimeoutSecs" : 10,
                "electionTimeoutMillis" : 10000,
                "catchUpTimeoutMillis" : -1,
                "catchUpTakeoverDelayMillis" : 30000,
                "getLastErrorModes" : {

                },
                "getLastErrorDefaults" : {
                        "w" : 1,
                        "wtimeout" : 0
                },
                "replicaSetId" : ObjectId("5a296e4da9fdf50c1fc967ae")
        }
}
>
```

Display the status of the replica set

To display dynamic information about the status of the replica set, use the `rs.status()` command instead:

```
> rs.status()
{
        "set" : "rs0",
        "date" : ISODate("YYYY-MM-DDTHH:MM:SS.sssZ"),
        "myState" : 1,
        "term" : NumberLong(1),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1512664965, 1),
                        "t" : NumberLong(1)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1512664965, 1),
                        "t" : NumberLong(1)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1512664965, 1),
                        "t" : NumberLong(1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1512664965, 1),
                        "t" : NumberLong(1)
                }
        },
        "members" : [
                {
                        "_id" : 0,
                        "name" : "casdev-srv01.newschool.edu:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 916,
                        "optime" : {
                                "ts" : Timestamp(1512664965, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("YYYY-MM-DDTHH:MM:SS
Z"),
                        "electionTime" : Timestamp(1512664653, 2),
                        "electionDate" : ISODate("2017-12-07T16:37:33
Z"),
                        "configVersion" : 3,
                        "self" : true
                },
                {
                        "_id" : 1,
```

```
                        "name" : "casdev-srv02.newschool.edu:27017",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 247,
                        "optime" : {
                                "ts" : Timestamp(1512664965, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1512664965, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("YYYY-MM-DDTHH:MM:SS
Z"),
                        "optimeDurableDate" : ISODate("YYYY-MM-DDTH
H:MM:SSZ"),
                        "lastHeartbeat" : ISODate("YYYY-MM-DDTHH:MM:S
S.sssZ"),
                        "lastHeartbeatRecv" : ISODate("YYYY-MM-DDTH
H:MM:SS.sssZ"),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "casdev-srv01.newschool.edu:270
17",
                        "configVersion" : 3
                },
                {
                        "_id" : 2,
                        "name" : "casdev-srv03.newschool.edu:27017",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 99,
                        "optime" : {
                                "ts" : Timestamp(1512664965, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1512664965, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("YYYY-MM-DDTHH:MM:SS
Z"),
                        "optimeDurableDate" : ISODate("YYYY-MM-DDTH
H:MM:SSZ"),
                        "lastHeartbeat" : ISODate("YYYY-MM-DDTHH:MM:S
S.sssZ"),
```

```
                        "lastHeartbeatRecv" : ISODate("YYYY-MM-DDTH
H:MM:SS.sssZ"),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "casdev-srv01.newschool.edu:270
17",
                        "configVersion" : 3
                }
        ],
        "ok" : 1,
        "operationTime" : Timestamp(1512664965, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1512664965, 1),
                "signature" : {
                        "hash" : BinData(0,"wGZmpqOqx1Xz1XDrsa2129JA
d+c="),
                        "keyId" : NumberLong("6496845223040122881")
                }
        }
}
>
```

The `members[n].stateStr` element indicates, for each member, whether it is the primary or a secondary member of the replica set.

Display current replication status

To display the current status of replicating data from the primary to the slave (secondary) servers, use the `rs.printSlaveReplicationInfo()` command:

```
> rs.printSlaveReplicationInfo()
source: casdev-srv02.newschool.edu:27017
        syncedTo: Ddd MMM DD YYYY HH:MM:DD GMT-0500 (EST)
        0 secs (0 hrs) behind the primary
source: casdev-srv03.newschool.edu:27017
        syncedTo: Ddd MMM DD YYYY HH:MM:DD GMT-0500 (EST)
        0 secs (0 hrs) behind the primary
>
```

Exit the `mongo` shell

Exit the `mongo` shell:

```
> exit
bye
```

References

- MongoDB: Deploy Replica Set With Keyfile Access Control
- Linode: Create a MongoDB Replica Set

Test the replica set

To verify that replication is working, we will create some test data on the primary member, and then try to read it from one (or more) of the secondary members.

Connect to the primary with the `mongo` shell

On the primary replicat set member (as determined by the output from `rs.staus()` in the previous section), start the `mongo` shell again by running the command

```
casdev-srv01# mongo -u mongoadmin -p --authenticationDatabase admin
MongoDB shell version v3.6.0
Enter password:
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.0
rs0:PRIMARY>
```

and entering the correct password ("`changeit`"). The `mongo` shell prompt now displays the replica set name and member status.

Create some test data

Enter the commands below to create a new database called `testDatabase` (databases are created automatically the first time they are used) and store some documents in a collection called `testCollection`:

```
rs0:PRIMARY> use testDatabase
switched to db testDatabase
rs0:PRIMARY> for (var i=1; i <= 10; i++) db.testCollection.insert( {
val: i } )
WriteResult({ "nInserted" : 1 })
rs0:PRIMARY>
```

Then retrieve the documents just created by running the command

```
rs0:PRIMARY> db.testCollection.find()
{ "_id" : ObjectId("5a298797050075eeef5df805"), "val" : 1 }
{ "_id" : ObjectId("5a298797050075eeef5df806"), "val" : 2 }
{ "_id" : ObjectId("5a298797050075eeef5df807"), "val" : 3 }
{ "_id" : ObjectId("5a298797050075eeef5df808"), "val" : 4 }
{ "_id" : ObjectId("5a298797050075eeef5df809"), "val" : 5 }
{ "_id" : ObjectId("5a298797050075eeef5df80a"), "val" : 6 }
{ "_id" : ObjectId("5a298797050075eeef5df80b"), "val" : 7 }
{ "_id" : ObjectId("5a298797050075eeef5df80c"), "val" : 8 }
{ "_id" : ObjectId("5a298797050075eeef5df80d"), "val" : 9 }
{ "_id" : ObjectId("5a298797050075eeef5df80e"), "val" : 10 }
rs0:PRIMARY>
```

Connect to a secondary with the `mongo` shell

Now connect with the `mongo` shell on one of the secondary members (e.g., ***casdev-srv02***) by running the command

```
casdev-srv02# mongo -u mongoadmin -p --authenticationDatabase admin
MongoDB shell version v3.6.0
Enter password:
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.0
rs0:SECONDARY>
```

and entering the correct password (“`changeit`”).

Enable secondary member read operations

By default, reads from secondary members are not allowed; this is to ensure that a query does not retrieve stale data. Even simple commands like `show dbs` and `show collections` will fail with an error. But in this case we want to read from the secondary, to make sure the data got copied from the primary. To enable this, run the command

```
rs0:SECONDARY> db.getMongo().setSlaveOk()
```

which will enable reading from the secondary for duration of this connection.

Check that everything was replicated

Run the commands

```
rs0:SECONDARY> show dbs
admin         0.000GB
config        0.000GB
local         0.000GB
testDatabase  0.000GB
rs0:SECONDARY> use testDatabase
switched to db testDatabase
rs0:SECONDARY> show collections
testCollection
rs0:SECONDARY>
```

to verify that `testDatabase` and `testCollection` are indeed available on the secondary. Then run the command

```
rs0:SECONDARY> db.testCollection.find()
{ "_id" : ObjectId("5a298797050075eeef5df805"), "val" : 1 }
{ "_id" : ObjectId("5a298797050075eeef5df807"), "val" : 3 }
{ "_id" : ObjectId("5a298797050075eeef5df808"), "val" : 4 }
{ "_id" : ObjectId("5a298797050075eeef5df806"), "val" : 2 }
{ "_id" : ObjectId("5a298797050075eeef5df80a"), "val" : 6 }
{ "_id" : ObjectId("5a298797050075eeef5df80c"), "val" : 8 }
{ "_id" : ObjectId("5a298797050075eeef5df80d"), "val" : 9 }
{ "_id" : ObjectId("5a298797050075eeef5df80b"), "val" : 7 }
{ "_id" : ObjectId("5a298797050075eeef5df80e"), "val" : 10 }
{ "_id" : ObjectId("5a298797050075eeef5df809"), "val" : 5 }
rs0:SECONDARY>
```

to check that, indeed, the data inserted on the primary is also present on the secondary. The data may not appear in numerical order, since the command did not attempt to sort them, but they should all be present.

Delete the test database

Exit the `mongo` shell on the secondaries, and then run the commands

```
rs0:PRIMARY> show dbs
admin         0.000GB
config        0.000GB
local         0.000GB
testDatabase  0.000GB
rs0:SECONDARY> use testDatabase
switched to db testDatabase
rs0:SECONDARY> db.dropDatabase()
{
        "dropped" : "testDatabase",
        "ok" : 1,
        "operationTime" : Timestamp(1512671689, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1512671689, 2),
                "signature" : {
                        "hash" : BinData(0,"CpHBrLKG56+ehaMf8Uk5QlzeS
X8="),
                        "keyId" : NumberLong("6496845223040122881")
                }
        }
}
rs0:PRIMARY>
```

on the primary to remove the test database and its contents.

References

- Linode: Create a MongoDB Replica Set

Configure MongoDB to use TLS/SSL

Communications between the CAS server and calling applications (clients) are protected by TLS/SSL to prevent the disclosure of users' security credentials and/or CAS ticket-granting tickets. Since MongoDB will be used to store ticket-granting tickets (and other sensitive information), communications between the CAS server and MongoDB should likewise be protected by TLS/SSL.

Generate private keys and certificate signing requests

Each of the replica set members (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) will need its own TLS/SSL certificate. Run the commands

```
casdev-srv01# cd /etc/pki/tls/private
casdev-srv01# openssl req -nodes -newkey rsa:2048 -sha256 -keyout cas
dev-srv01.key -out casdev-srv01.csr
Generating a 2048 bit RSA private key
.....................................................................
...........
.....................+++
.....................................................................
...........
..............+++
writing new private key to 'casdev-srv01.key'
-----
You are about to be asked to enter information that will be incorpora
ted
into your certificate request.
What you are about to enter is what is called a Distinguished Name o
r a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:New York
Locality Name (eg, city) [Default City]:New York
Organization Name (eg, company) [Default Company Ltd]:The New School
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:casdev-srv0
1.newschool.edu
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
casdev-srv01#
```

on ***casdev-srv01*** to generate a private key and certificate signing request. (Replace the contents of the Distinguished Name fields with values appropriate for your organization.) Repeat the commands on ***casdev-srv02*** and ***casdev-srv03*** to generate private keys and certificate signing requests on those servers as well, making the obvious host name substitutions on each server.

Submit all three certificate signing requests (`casdev-srvNN.csr`) to your certificate authority to obtain certificates. When the certificates come back from the certificate authority, copy the certificate for each server and any intermediate certificate(s) into `/etc/pki/tls/certs` on the applicable server, saving them as `casdev-srvNN.crt`,

`casdev-srvNN-intermediate.crt`, `casdev-srvNN-root.crt`, etc. (you may need to separately download the root certificate from the certificate authority's web site). If your certificate authority offers multiple certificate formats, opt for the PEM format, which looks like:

```
-----BEGIN CERTIFICATE-----
AQEFAAOCAQ8AMIIBCgKCAQEAtGCKiysqhQF4/AA5Pvi7EIIRqbtVx/IF0CAFK8lv
6uDJDHjd7bSNhhzYJxUNCdN0DacYT5wI/s4n3mLEXQrIt0KsUdPD+s7qP9Lw05hI
WaG7KhP6RZ+UtWSvHwIZJUHvlJvh2GlARw/XwV3iHG3mxfl5nCLNihAR9S1r2qEY
...several more lines of base64-encoded data...
-----END CERTIFICATE----
```

Combine the certificate and private key into a single `.pem` file

The MongoDB server requires the certificate and the private key to be stored in a single file. Run the commands

```
casdev-srv01# cd /etc/pki/tls
casdev-srv01# cat private/casdev-srv01.key certs/casdev-srv01.crt >
/var/lib/mongo/mongod-cert.pem
casdev-srv01# chown mongod.mongod /var/lib/mongo/mongod-cert.pem
casdev-srv01# chmod 400 /var/lib/mongo/mongod-cert.pem
```

on ***casdev-srv01*** to create the combined file in `/var/lib/mongo/mongod-cert.pem`. Repeat these commands on ***casdev-srv02*** and ***casdev-srv03***.

Combine the root and intermediate certificates into a single `.pem` file

The MongoDB server also requires that the certificate chain (the intermediate certificate(s) and the root certificate) from the certificate authority be provided in a single file. Run the commands

```
casdev-srv01# cd /etc/pki/tls
casdev-srv01# cat certs/casdev-srv01-intermediate.crt casdev-srv01-ro
ot.crt > /var/lib/mongo/mongod-cafile.pem
casdev-srv01# chown mongod.mongod /var/lib/mongo/mongod-cafile.pem
casdev-srv01# chmod 400 /var/lib/mongo/mongod-cafile.pem
```

on ***casdev-srv01*** to create the combined file in `/var/lib/mongo/mongod-cafile.pem`. Repeat these commands on ***casdev-srv02*** and ***casdev-srv03***.

Update the MongoDB configuration file

Edit the `/etc/mongod.conf` file on ***casdev-master*** and add an `ssl` subsection to the `net` section, as shown below:

```
net:
  port: 27017
  bindIp: 0.0.0.0
  ssl:
    mode: requireSSL
    allowConnectionsWithoutCertificates: true
    PEMKeyFile: /var/lib/mongo/mongod-cert.pem
    CAFile: /var/lib/mongo/mongod-cafile.pem
```

Then run the commands

```
casdev-master# for i in 01 02 03
> do
> scp -p /etc/mongod.conf casdev-srv${i}:/etc/mongod.conf
> ssh casdev-srv${i} "systemctl start mongod-disable-thp; systemctl r
estart mongod"
> done
mongod.conf                                   100%  813    41.2KB/
s   00:00
mongod.conf                                   100%  813    53.4KB/
s   00:00
mongod.conf                                   100%  813   721.3KB/
s   00:00
casdev-master#
```

to copy the updated configuration file to each member of the replica set and (re)start the `mongod` server.

Test the TLS/SSL configuration

To test the TLS/SSL configuration, use the `mongo` shell with the `--ssl` option to verify that you can connect to the server and execute a command. Run the commands

```
casdev-master# mongo -u mongoadmin -p --authenticationDatabase admin
--ssl --host rs0/casdev-srv01.newschool.edu,casdev-srv02.newschool.ed
u,casdev-srv03.newschool.edu
MongoDB shell version v3.6.0
Enter password:
connecting to: mongodb://casdev-srv01.newschool.edu:27017,casdev-srv0
2.newschool.edu:27017,casdev-srv03.newschool.edu:27017/?replicaSet=rs
0
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Starting new replic
a set monitor for rs0/casdev-srv01.newschool.edu:27017,casdev-srv02.n
ewschool.edu:27017,casdev-srv03.newschool.edu:27017
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Successfully connec
ted to casdev-srv02.newschool.edu:27017 (1 connections now open to ca
sdev-srv02.newschool.edu:27017 with a 5 second timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [ReplicaSetMonitor-TaskExecut
or-0] Successfully connected to casdev-srv01.newschool.edu:27017 (1 c
onnections now open to casdev-srv01.newschool.edu:27017 with a 5 seco
nd timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Successfully connec
ted to casdev-srv03.newschool.edu:27017 (1 connections now open to ca
sdev-srv03.newschool.edu:27017 with a 5 second timeout)
MongoDB server version: 3.6.0
rs0:PRIMARY> show dbs
admin   0.000GB
casdb   0.000GB
config  0.000GB
local   0.014GB
rs0:PRIMARY> exit
bye
casdev-master#
```

Note that, now that a replica set has been established, the `--host` option must be provided, listing the replica set name and the names of all the replica set members, rather than just letting `mongo` connect to the local host. This will ensure that the shell connects to the primary replica set member, regardless of which member that happens to be at the moment.

References

- MongoDB: Configure mongod and mongos for TLS/SSL

Create the CAS database and user

We will create a CAS-specific database where the CAS server can store all its data. Each of the different modules (ticket registry, service registry, etc.) will store its information in a separate collection within this database. We will also create a "regular" user (one that does not have administrative rights) to be used by the CAS servers to access these tables.

Create the CAS database

MongoDB does not have a special command to create a database. Rather, the database is created the first time it is used. To create the database, connect to the primary replica set member with the `mongo` shell and issue a `use <databasename>` command:

```
casdev-master# mongo -u mongoadmin -p --authenticationDatabase admin
--ssl --host rs0/casdev-srv01.newschool.edu,casdev-srv02.newschool.ed
u,casdev-srv03.newschool.edu
MongoDB shell version v3.6.0
Enter password:
connecting to: mongodb://casdev-srv01.newschool.edu:27017,casdev-srv0
2.newschool.edu:27017,casdev-srv03.newschool.edu:27017/?replicaSet=rs
0
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Starting new replic
a set monitor for rs0/casdev-srv01.newschool.edu:27017,casdev-srv02.n
ewschool.edu:27017,casdev-srv03.newschool.edu:27017
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Successfully connec
ted to casdev-srv02.newschool.edu:27017 (1 connections now open to ca
sdev-srv02.newschool.edu:27017 with a 5 second timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [ReplicaSetMonitor-TaskExecut
or-0] Successfully connected to casdev-srv01.newschool.edu:27017 (1 c
onnections now open to casdev-srv01.newschool.edu:27017 with a 5 seco
nd timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Successfully connec
ted to casdev-srv03.newschool.edu:27017 (1 connections now open to ca
sdev-srv03.newschool.edu:27017 with a 5 second timeout)
MongoDB server version: 3.6.0
rs0:PRIMARY> use casdb
switched to db casdb
rs0:PRIMARY>
```

This will create a database called `casdb`.

Create a database user

Database users can be created in the `admin` database, or in the database they will be accessing. If the user is in a different database than the one being connected to however, then the connection command must specify the database to authenticate against. To simplify things, the CAS database user will be created in the `casdb` database created above. In the `mongo` shell, switch to the `casdb` database and create a new user by running the commands

```
rs0:PRIMARY> use casdb
switched to db casdb
rs0:PRIMARY> db.createUser( { user: "mongocas", pwd: "changeit", role
s: [ { role: "readWrite", db: "casdb" } ] } )
Successfully added user: {
        "user" : "mongocas",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "casdb"
                }
        ]
}
rs0:PRIMARY>
```

This will create a user named `mongocas` with password `changeit`. The user will have read/write access to the `casdb` database, and no access to any other database (this can be changed later, by adjusting the user's roles).

> ❗ **Warning:** The command above uses `changeit` as the value of the `mongocas` password. Obviously, something other than this should be used in a production MongoDB deployment.

Test the new database and user

Test the new database and user by exiting the administrative `mongo` shell and running the command

```
rs0:PRIMARY> exit
bye
casdev-master# mongo casdb -u mongocas -p --ssl --host rs0/casdev-srv
01.newschool.edu,casdev-srv02.newschool.edu,casdev-srv03.newschool.ed
u
MongoDB shell version v3.6.0
Enter password:
connecting to: mongodb://casdev-srv01.newschool.edu:27017,casdev-srv0
2.newschool.edu:27017,casdev-srv03.newschool.edu:27017/casdb?replicaS
et=rs0
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Starting new replic
a set monitor for rs0/casdev-srv01.newschool.edu:27017,casdev-srv02.n
ewschool.edu:27017,casdev-srv03.newschool.edu:27017
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Successfully connec
ted to casdev-srv02.newschool.edu:27017 (1 connections now open to ca
sdev-srv02.newschool.edu:27017 with a 5 second timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [ReplicaSetMonitor-TaskExecut
or-0] Successfully connected to casdev-srv01.newschool.edu:27017 (1 c
onnections now open to casdev-srv01.newschool.edu:27017 with a 5 seco
nd timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Successfully connec
ted to casdev-srv03.newschool.edu:27017 (1 connections now open to ca
sdev-srv03.newschool.edu:27017 with a 5 second timeout)
MongoDB server version: 3.6.0
rs0:PRIMARY> exit
bye
casdev-master#
```

to connect to the `casdb` database as the `mongocas` user.

References

- MongoDB: Add Users

Setting up the ticket registry

Summary: A distributed ticket registry, accessible to all CAS servers in the environment, will be used to ensure that tickets can be located (and validated) by any server in the environment, not just the server that created it.

To support high availability/fault tolerance, the CAS server environment has been built with a pool of servers behind a load balancer. However, as configured up to this point, each server in the pool behaves as an independent entity. This is immediately obvious when trying to access the "secure" content on either ***casdev-casapp*** or ***casdev-samlsp*** when more than one of the servers in the pool is up and running:

1. When a user attempts to access a CAS-protected application (or CAS-protected content within the application), the application checks to see if the user has provided a CAS Service Ticket (ST) as authorization. If an ST has not been provided (as happens when the application is first accessed), the application sends the user (usually with a web browser redirect) to the CAS server to obtain one.
2. When the user accesses the CAS server, the load balancer (which holds the IP address registered to the CAS server's public DNS host record) will connect the user to one of the serves in the pool using whatever load balancing strategy has been configured (e.g., round-robin).
3. The CAS server will query the user's client (usually by looking for a web browser cookie) to see if the user has a CAS Ticket Granting Ticket (TGT):
    a. If the user does not have a TGT, then CAS will prompt the user to enter his or her credentials (username, password, multi-factor authentication, etc.) and, upon successful authentication, create a TGT for the user.
    b. If the user already has a TGT, then CAS will attempt to validate it (make sure it can be decrypted, hasn't expired, etc.) and, if validation is successful, allow the user to proceed without having to enter his or her credentials again. **Problem:** By default, the CAS server stores the information it needs to validate TGTs (the *ticket registry*) in memory. Thus, if one server in the pool created a TGT, another server in the pool will be unable to validate it, because it doesn't have access to that information.
4. Once the TGT has been created/validated, the CAS server uses it to create an ST for the application, and sends the user (again, usually with a

web browser redirect) back to the application.

5. The application takes the ST provided by the user and sends it via a back channel (user-transparent) communication to the CAS server to be validated. The CAS server will use the TGT and other information to validate the ST, and return the results to the application. This is also the point at which any user attributes (e.g., from Active Directory or LDAP) are returned to the application. **Problem:** The CAS server needs information from the ticket registry to validate STs. Since, as mentioned above, this information is stored in memory by default, CAS servers cannot validate STs created by other servers in the pool because they do not have access to the necessary information.
6. Once the ST has been validated, the application allows the user to access the protected content.

The problems identified in steps 3(b) and 5 above are why, when testing the CAS client and SAML client in previous sections, we always shut down all but one of the CAS servers in the pool. (For more details on the steps above, including flow diagrams, see CAS Protocol .)

To solve the problem of each server in the pool not having the information needed to validate TGTs and STs created by other servers, we will replace the default in-memory ticket registry with one stored in MongoDB. Each CAS server in the pool will save its tickets to the database, and any other server in the pool will be able to access the tickets created by other servers when necessary.

Update the server configuration

Enabling the MongoDB ticket registry requires adding a new dependency to the Maven project object model, rebuilding the server, and updating `cas.properties`.

Add the dependency to the project model

To add the MongoDB ticket registry to the CAS server, edit the file `pom.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the dependencies section (around line 69), which should look something like this:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml-idp</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-duo</artifactId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

Insert the new dependency at the bottom of the section:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml-idp</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-duo</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-mongo-ticket-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

This will instruct Maven to download the appropriate code modules and build them into the server.

Rebuild the server

Run Maven to rebuild the server according to the new model:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 17.257 s
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 39M/97M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

Configure MongoDB ticket registry properties

Add the following settings to `etc/cas/config/cas.properties` in the `cas-overlay-template` directory:

```
#
# Components of the MongoDB connection string broken out for ease of
editing.
# See https://docs.mongodb.com/reference/connection-string/
#
mongo.db:                               casdb
mongo.rs:                               rs0
mongo.opts:                             &ssl=false
mongo.creds:                            mongocas:changeit
mongo.hosts:                            casdev-srv01.newschool.edu,cas
dev-srv02.newschool.edu,casdev-srv03.newschool.edu

#
# The connection string, assembled
#
mongo.uri:                              mongodb://${mongo.creds}@${mon
go.hosts}/${mongo.db}?replicaSet=${mongo.rs}${mongo.opts}

#
# Ticket registry
#
cas.ticket.registry.mongo.clientUri:    ${mongo.uri}
```

The `cas.ticket.registry.mongo.clientUri` property is the only setting specific to the ticket registry that needs to be set; it takes a MongoDB connection string as a value. The format of a MongoDB connection string is:

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,host
N[:portN]]][/[database][?options]]
```

The `mongo.*` properties are optional; they make it easier to edit the components of the connection string, and will also make it easier to repeat the connection string in other settings (such as for the [MongoDB service registry][high-avail_service-registry_overview]) without having to duplicate values that may change over time.

Enable MongoDB Java driver logging

The Log4J configuration file included with the Maven WAR overlay template does not include a logger for the MongoDB Java driver, which is what the CAS server uses to communicate with MongoDB. To enable MongoDB Java driver logging, edit the file `etc/cas/config/log4j2.xml` on the master build server (***casdev-master***) and locate the bottom of the list of `AsyncLogger` definitions (around line 97), which should look something like this:

```
<AsyncLogger name="org.ldaptive" level="warn" />
<AsyncLogger name="com.hazelcast" level="warn" />
<AsyncLogger name="org.jasig.spring" level="warn" />

<!-- Log perf stats only to perfStats.log -->
```

Add a new definition for `org.mongodb.driver` to the list:

```
<AsyncLogger name="org.ldaptive" level="warn" />
<AsyncLogger name="com.hazelcast" level="warn" />
<AsyncLogger name="org.jasig.spring" level="warn" />
<AsyncLogger name="org.mongodb.driver" level="warn" />

<!-- Log perf stats only to perfStats.log -->
```

Although this line is not critical to the regular operation of the CAS server (warnings and errors will just propagate upward), it is immensely helpful when trying to debug MongoDB connection problems (set `level` to `debug`).

References

- CAS 5: MongoDB Ticket Registry
- CAS 5: Configuration Properties: MongoDB Ticket Registry
- MongoDB: Connection String URI Format

Install and test the application

The MongoDB ticket registry can be tested by installing the updated CAS software on the CAS servers and restarting it, and then accessing the test clients from multiple browsers.

Install and test on the master build server

Use the updated build and installation scripts (page 248) (or repeat the commands) to install the updated CAS server on the master build server (***casdev-master***) and restart Tomcat:

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors. Then connect to MongoDB with the `mongo` shell and check to see that the ticket registry collections have been created. Use the MongoDB connection string to ensure that you get connected to the primary:

```
casdev-master# mongo 'mongodb://mongocas:changeit@casdev-srv01.newsch
ool.edu,casdev-srv02.newschool.edu,casdev-srv03.newschool.edu/casdb?r
eplicaSet=rs0&ssl=false'
MongoDB shell version v3.6.0
connecting to: mongodb://mongocas:changeit@casdev-srv01.newschool.ed
u,casdev-srv02.newschool.edu,casdev-srv03.newschool.edu/casdb?replica
Set=rs0&ssl=false
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Starting new replic
a set monitor for rs0/casdev-srv01.newschool.edu:27017,casdev-srv02.n
ewschool.edu:27017,casdev-srv03.newschool.edu:27017
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [ReplicaSetMonitor-TaskExecut
or-0] Successfully connected to casdev-srv01.newschool.edu:27017 (1 c
onnections now open to casdev-srv01.newschool.edu:27017 with a 5 seco
nd timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Successfully connec
ted to casdev-srv02.newschool.edu:27017 (1 connections now open to ca
sdev-srv02.newschool.edu:27017 with a 5 second timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [ReplicaSetMonitor-TaskExecut
or-0] Successfully connected to casdev-srv03.newschool.edu:27017 (1 c
onnections now open to casdev-srv03.newschool.edu:27017 with a 5 seco
nd timeout)
MongoDB server version: 3.6.0
rs0:PRIMARY> show collections
proxyGrantingTicketsCollection
proxyTicketsCollection
samlArtifactsCache
samlAttributeQueryCache
serviceTicketsCollection
ticketGrantingTicketsCollection
rs0:PRIMARY>
```

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the updated build and installation scripts (page 248):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Test the operation of the registry

To test the operation of the registry, start a terminal session on each of the CAS servers (***casdev-srv01***, ***casdev-srv02***, and ***casdev-srv03***) and run the command

```
# tail -f /var/log/cas/cas.log
```

Then:

1. Start a web browser and access the CAS client application (***casdev-casapp***). Click the link to access the secured content and log into CAS. Watch the terminal windows to see which CAS server processes the request. Note that it is quite likely (although not certain) that one server will handle the authnetication from your browser, and another server will handle the ticket validation from the client application.
2. Using the same web browser, access the SAML client application (***casdev-samlsp***). Again, click the link to access the secured content and watch the terminal windows to see which CAS server processes the request. There should be no need to re-enter your username and password, since the server should find your ticket granting ticket in the database.
3. Start a different web browser (or use a different computer) and repeat steps 1 and 2.
4. Start a different (from either of the first two) web browser, but this time access the status dashboard (`https://casdev.newschool.edu/cas/status/dashboard`). On the dashboard, click on the **SSO Sessions** button to see a list of all the current sessions (there should be one line per user that you have authenticated with, and the “Usage Count” column should show the number of services each user has authenticated to).
5. In the `mongo` shell, run the command

```
rs0:PRIMARY> db.ticketGrantingTicketsCollection.distinct("ticketId")
[
        "TGT-1--H-sl9Yzq9a5gJYi1bsoV6ai_kukv7iD4_njJzuANUwZo6qosuX2
r_3U-oxD5K0LBBg-casdev-srv01",
        "TGT-1-Blp3zphQTS-CS4JWf4Tb4u7c1Pl5i5TpK11f8-Eu5Sh-gAgjVi_KbR
U1pqgQDYLLRL0-casdev-srv03",
        "TGT-1-KFfoL9mnLFMXx5il2Qyj-vMEN_3i5-i1dHBFoGuacr0COgybq-m5V
K-E8k-ljXwoQMk-casdev-srv02"
]
rs0:PRIMARY>
```

to see that all of the tickets have been created in the database. Note that the end of each ticket identifier contains the host name of the particular CAS server that created it.

Commit changes to Git

Before moving on to the next task, commit the changes made to `pom.xml`, `cas.properties`, and `log4j2.xml` to Git to make changes easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git add etc/cas/config/log4j2.xml
casdev-master# git add pom.xml
casdev-master# git commit -m "Added MongoDB ticket registry"
[newschool-casdev 57302ae] Added MongoDB ticket registry
 3 files changed, 33 insertions(+), 4 deletions(-)
casdev-master#
```

on the master build server (***casdev-master***).

Setting up the service registry

Summary: A distributed service registry, accessible to all CAS servers, will be used to ensure that every server has the most up-to-date information about authorized services, and to allow the registry to be maintained from a single administration point.

The JSON service registry (page 107) works well in a single server environment. But in an environment with a pool of servers, it doesn't. Most significantly, the service management webapp (page 236) won't work correctly, because any modifications it makes to the service registry will only take effect on the particular pool server where the webapp session is running. The other servers' registries will be out of date until some out-of-band process (manual or automated) can update them with the new information. That update process however is complicated by the fact that the service management webapp runs on every CAS server (for high availability/fault tolerance), so different changes can be made on different servers, requiring the synchronization process to be capable of performing N-way merges and resolving any resultant conflicts.

To solve this problem, we will replace the JSON service registry with one stored in MongoDB. Each CAS server in the pool will load (and reload) its list of authorized services from the database and every instance of the service management webapp will write to the database, thus ensuring that all servers in the pool always have up-to-date, identical information about authorized services.

Update the server configuration

Enabling the MongoDB service registry requires updating a dependency in the project object model, rebuilding the server, and updating configuration properties. The same steps must be performed on the management webapp, which also works with the service registry.

Update the dependency in the project model

As with all other features of the CAS server (and the management webapp), the MongoDB service registry is enabled by adding a new dependency to the project object model (`pom.xml`). But this time we are actually replacing one feature (the JSON service registry) with another, so rather than adding a new dependency to the list, we will update one in place.

CAS server

Edit the file `pom.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the `cas-server-support-json-service-registry` dependency (around line 79) that was added when the service registry was first enabled (page 108)], which should look like this:

```
<dependencies>
    ....
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-json-service-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
    ....
</dependencies>
```

Replace the `json` in the dependency name with `mongo` , so that it now looks like this:

```
<dependencies>
    ....
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-mongo-service-registry</artifa
ctId>
        <version>${cas.version}</version>
    </dependency>
    ....
</dependencies>
```

This will instruct Maven to download the appropriate code modules and build them into the server.

Service management webapp

Edit the file `pom.xml` in the `cas-management-overlay` directory on the master build server (***casdev-master***) and locate the `cas-server-support-json-service-registry` dependency (around line 78) that was added when the management webapp was first built (page 240), and repeat the change described above (replace `json` with `mongo` in the name of the dependency).

Rebuild the application

Run Maven to rebuild the applications according to the new models.

CAS server

From the `cas-overlay-template` directory:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 25.411 s
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 37M/96M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

Manaagement webapp

From the `cas-management-overlay` directory:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 8.229 s
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 30M/72M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

Configure MongoDB service registry properties

Both the CAS server and the management webapp need to be configured to recognize and use the MongoDB service registry.

CAS server

Add the following settings to `etc/cas/config/cas.properties` in the `cas-overlay-template` directory:

```
#
# Components of the MongoDB connection string broken out for ease of
editing.
# See https://docs.mongodb.com/manual/reference/connection-string/
#
mongo.db:                               casdb
mongo.rs:                               rs0
mongo.opts:                             &ssl=false
mongo.creds:                            mongocas:changeit
mongo.hosts:                            casdev-srv01-lid.newschool.ed
u,casdev-srv02-lid.newschool.edu,casdev-srv03-lid.newschool.edu

#
# The connection string, assembled
#
mongo.uri:                              mongodb://${mongo.creds}@${mo
ngo.hosts}/${mongo.db}?replicaSet=${mongo.rs}${mongo.opts}

#
# Service registry
#
cas.serviceRegistry.mongo.clientUri:    ${mongo.uri}
cas.serviceRegistry.mongo.collection:   casServiceRegistry
```

The `cas.serviceRegistry.mongo.clientUri` property is the only setting specific to the service registry that needs to be set; it takes a MongoDB connection string as a value. The format of a MongoDB connection string is:

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,host
N[:portN]]][/[database][?options]]
```

The `mongo.*` properties are optional; they make it easier to edit the components of the connection string, and also make it easier to repeat the connection string in other settings (such as for the MongoDB ticket registry (page 361)) without having to duplicate values that may change over time.

The `cas.serviceRegistry.mongo.collection` property setting is not required, but is recommended. By default, CAS calls the collection `cas-service-registry`. This is a valid MongoDB collection name, but the `mongo` shell does not accept collection names that contain hyphens in some commands. Although it's possible to work around that using alternative command syntax, it's easier to just avoid the problem altogether by using a collection name that doesn't contain hyphens.

Do not delete the `cas.serviceRegistry.json.location` property from `cas.properties` just yet; it's still needed to transfer the contents of the JSON service registry to the MongoDB service registry.

Management webapp

Copy the settings that were added to the CAS server above to `etc/cas/config/management.properties` in the `cas-management-overlay` directory.

Delete the `cas.serviceRegistry.json.location` property from `management.properties`; it is not needed now that we are using the MongoDB service registry.

References

- CAS 5: Mongo Service Registry
- CAS 5: Configuration Properties: MongoDb Service Registry

Load the MongoDB service registry from the JSON service registry

If no other service registry is configured, CAS will use an in-memory service registry (not suitable for production deployments) and, to make it possible to experiment with the server, automatically initialize that registry from some default JSON service definitions included with the software (these are stored in the application classpath). But the auto-initialization functionality is actually more flexible than this: it can initialize any configured service registry, not just the in-memory service registry, and it can be told where to look for the set of JSON service definitions that it will populate into that registry. We will make use of these features to transfer the contents of the existing JSON service registry to the new MongoDB service registry.

Configure service registry auto-initialization

Edit the file `etc/cas/config/cas.properties` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the `cas.serviceRegistry.json.location` property (around line 7) that was added when we set up the original service registry (page 110):

```
cas.serviceRegistry.json.location:     file:/etc/cas/services
```

(If you deleted this setting when adding the MongoDB settings in the previous section, add it back, because it's needed for this step.) Add the `cas.serviceRegistry.initFromJson` property to enable the automatice service registry initialization functionality:

```
cas.serviceRegistry.json.location:     file:/etc/cas/services
cas.serviceRegistry.initFromJson:      true
```

Install and run on the master build server

Use the updated build and installation scripts (page 248) (or repeat the commands) to install the updated CAS server and management webapp on the master build server (***casdev-master***) and restart Tomcat:

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors. This warning may appear in `cas.log` :

```
Runtime memory is used as the persistence storage for retrieving and
persisting service definitions. Changes that
are made to service definitions during runtime WILL be LOST when the
web server is restarted. Ideally for
production, you need to choose a storage option (JDBC, etc) to store
and track service definitions.
```

and can be safely ignored (its appearance is a side effect caused by the fact that we haven't actually created anything in the MongoDB service registry yet).

Verify that the MongoDB service registry was created and populated

Once the server has finished its startup, connect to MongoDB with the `mongo` shell and check to see that the service registry collection (`casServiceRegistry`) has been created, and that it has the appropriate contents. Use the MongoDB connection string to ensure that you get connected to the primary:

```
casdev-master# mongo 'mongodb://mongocas:changeit@casdev-srv01.newsch
ool.edu,casdev-srv02.newschool.edu,casdev-srv03.newschool.edu/casdb?r
eplicaSet=rs0&ssl=false'
MongoDB shell version v3.6.0
connecting to: mongodb://mongocas:changeit@casdev-srv01.newschool.ed
u,casdev-srv02.newschool.edu,casdev-srv03.newschool.edu/casdb?replica
Set=rs0&ssl=false
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Starting new replic
a set monitor for rs0/casdev-srv01.newschool.edu:27017,casdev-srv02.n
ewschool.edu:27017,casdev-srv03.newschool.edu:27017
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [ReplicaSetMonitor-TaskExecut
or-0] Successfully connected to casdev-srv01.newschool.edu:27017 (1 c
onnections now open to casdev-srv01.newschool.edu:27017 with a 5 seco
nd timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [thread1] Successfully connec
ted to casdev-srv02.newschool.edu:27017 (1 connections now open to ca
sdev-srv02.newschool.edu:27017 with a 5 second timeout)
YYYY-MM-DDTHH:MM:SS.sss-0000 I NETWORK  [ReplicaSetMonitor-TaskExecut
or-0] Successfully connected to casdev-srv03.newschool.edu:27017 (1 c
onnections now open to casdev-srv03.newschool.edu:27017 with a 5 seco
nd timeout)
MongoDB server version: 3.6.0
rs0:PRIMARY> show collections
casServiceRegistry
proxyGrantingTicketsCollection
proxyTicketsCollection
samlArtifactsCache
samlAttributeQueryCache
serviceTicketsCollection
ticketGrantingTicketsCollection
rs0:PRIMARY> db.casServiceRegistry.distinct("serviceId")
[
        "https://casdev.newschool.edu/cas/idp/profile/SAML2/Callbac
k.+",
        "^https://casdev-casapp.newschool.edu/secured-by-ca
s(\\z|/.*)",
        "https://casdev-samlsp.newschool.edu/shibboleth",
        "^https://casdev-casapp.newschool.edu/return-mappe
d(\\z|/.*)",
        "^https://casdev-casapp.newschool.edu/secured-by-cas-du
o(\\z|/.*)",
        "^https://casdev.newschool.edu/cas/status/dashboar
d(\\z|/.*)",
        "^https://casdev.newschool.edu/cas-management(\\z|/.*)"
]
rs0:PRIMARY>
```

There should be one entry in the collection for each service defined in the JSON service registry.

Shut down the application

Once the MongDB service registry has been initialized, shut down the CAS server by running the command

```
casdev-master# systemctl stop tomcat
```

Remove service registry auto-initialization settings

Edit the file `etc/cas/config/cas.properties` in the `cas-overlay-template` directory again and delete the `cas.serviceRegistry.json.location` and `cas.serviceRegistry.initFromJson` settings.

Install and test the application

The MongoDB service registry can be tested by installing the updated CAS software on the CAS servers and restarting it, and then accessing the test clients and the management webapp.

Delete the JSON service registry

Before installing, delete the JSON service registry on the master build server and the CAS servers by either renaming or removing the `/etc/cas/services` directory:

```
casdev-master# mv /etc/cas/services /etc/cas/services.off
casdev-master# for i in 01 02 03
> do
> ssh casdev-srv${i} rm -rf /etc/cas/services
> done
casdev-master#
```

Install and test on the master build server

Use the updated build and installation scripts (page 248) (or repeat the commands) to install the updated CAS server on the master build server (***casdev-master***) and restart Tomcat:

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the updated build and installation scripts (page 248):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Test the operation of the registry

To test the operation of the registry, log into the management webapp and verify that all expected services are present, and that a service can be added to the registry (and removed from it). You can also use the CAS and SAML client applications to ensure that you can access the protected areas of those applications and that all expected attributes are still being released.

Commit changes to Git

Before moving on to the next task, commit the changes made to `pom.xml` and `cas.properties` in the `cas-overlay-template` directory to Git to make changes easier to keep track of (and to enable reverting to earlier configurations easier). Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# git add etc/cas/config/cas.properties
casdev-master# git add pom.xml
casdev-master# git commit -m "Added MongoDB service registry"
[newschool-casdev bfc6d29] Added MongoDB service registry
 2 files changed, 7 insertions(+), 3 deletions(-)
casdev-master#
```

on the master build server (***casdev-master***). Then do the same with `pom.xml` and `management.properties` in the `cas-services-management-overlay` directory:

```
casdev-master# cd /opt/workspace/cas-services-management-overlay
casdev-master# git add etc/cas/config/management.properties
casdev-master# git add pom.xml
casdev-master# git commit -m "Added MongoDB service registry"
[newschool-casdev-sm 7098c1f] Added MongoDB service registry
 2 files changed, 29 insertions(+), 4 deletions(-)
casdev-master#
```

Setting up distributed SAML metadata storage

Summary: Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla sodales turpis eu sem fermentum eleifend. Aliquam sodales dignissim lectus id condimentum. Phasellus sed dictum erat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nullam at tortor ac tellus interdum tincidunt et vitae est. Vivamus scelerisque sem diam. Nam orci mauris, fringilla ut magna ac, scelerisque fringilla tellus. Aliquam viverra, est ut tincidunt rutrum, orci nisl interdum lectus, ut gravida risus ipsum non tellus.

Setting up distributed configuration properties

Summary: Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla sodales turpis eu sem fermentum eleifend. Aliquam sodales dignissim lectus id condimentum. Phasellus sed dictum erat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nullam at tortor ac tellus interdum tincidunt et vitae est. Vivamus scelerisque sem diam. Nam orci mauris, fringilla ut magna ac, scelerisque fringilla tellus. Aliquam viverra, est ut tincidunt rutrum, orci nisl interdum lectus, ut gravida risus ipsum non tellus.

Google Apps (G Suite) integration

Summary: Google Apps (G Suite) can be configured to use CAS as a SAML2 IdP for single sign-on.

Google Apps (G Suite) allows an external SAML2 identity provider (IdP) to be used for user authentication (single sign-on) as an alternative to storing user passwords on Google's servers. Support for Google's specific SAML2 implementation was originally added to CAS in the CAS 3.*x* days, before CAS included general support for the SAML2 protocol. This code has been brought forward to CAS 5, and so integrating Google Apps is still done separately (and differently) from integrating other SAML2 services.

Add the Google Apps dependency to the project object model

To add Google Apps support to the CAS server, edit the `pom.xml` in the `cas-overlay-template` directory on the master build server (***casdev-master***) and locate the dependencies section (around line 69), which should look something like this:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-mongo-service-registry</artifa
ctId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml-idp</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-duo</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-mongo-ticket-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

Insert a dependency for `cas-server-support-saml-googleapps` below the one for `cas-server-support-saml-idp`:

```
<dependencies>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-webapp${app.server}</artifactId>
        <version>${cas.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-mongo-service-registry</artifa
ctId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-ldap</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml-idp</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-saml-googleapps</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-duo</artifactId>
        <version>${cas.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apereo.cas</groupId>
        <artifactId>cas-server-support-mongo-ticket-registry</artifac
tId>
        <version>${cas.version}</version>
    </dependency>
</dependencies>
```

This will instruct Maven to download the appropriate code modules and build them into the server.

Rebuild the server

Run Maven to rebuild the server according to the new model:

```
casdev-master# ./mvnw clean package
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
---
[INFO] Building cas-overlay 1.0
[INFO]
------------------------------------------------------------------------
---
(lots of diagnostic output... check for errors)
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
---
[INFO] Total time: 17.257 s
[INFO] Finished at: YYYY-MM-DDTHH:MM:SS-00:00
[INFO] Final Memory: 39M/97M
[INFO]
------------------------------------------------------------------------
---
casdev-master#
```

References

- CAS 5: Google Apps Integration

Generate keys and certificates

Google's SAML2 implementation requires that the SAML assertions exchanged with the CAS server be encrypted and signed. Therefore, it's necessary to generate a public/private key pair and an X.509 certificate for this purpose.

Use OpenSSL to generate the keys and certificate

Although it's not the only method, OpenSSL is perhaps the easiest way to create the keys and certificate. Run the commands

```
casdev-master# cd /opt/workspace/cas-overlay-template
casdev-master# mkdir etc/cas/google
casdev-master# cd etc/cas/google
casdev-master# openssl genrsa -out privatekey.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................
+++
.....................................................................
.................+++
e is 65537 (0x10001)
casdev-master# openssl rsa -in privatekey.pem -pubout -outform DER -o
ut publickey.der
writing RSA key
casdev-master# openssl pkcs8 -topk8 -inform PEM -outform DER -in priv
atekey.pem -out privatekey.der -nocrypt
casdev-master# openssl req -new -x509 -key privatekey.pem -out x509.p
em
You are about to be asked to enter information that will be incorpora
ted
into your certificate request.
What you are about to enter is what is called a Distinguished Name o
r a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:New York
Locality Name (eg, city) [Default City]:New York
Organization Name (eg, company) [Default Company Ltd]:The New School
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:newschool.ed
u
Email Address []:
```

Note: The CAS documentation provides slightly outdated instructions for creating the keys. To ensure compatibility, follow the directions from Google's documentation (linked below). The instructions above are based on the Google instructions.

When providing the Distinguished Name information for the X.509 certificate, use the same information you use when creating an SSL certificate. For the Common Name field, use the domain name of the Google Apps instance.

Although it's possible to include the keys and certificate in the CAS WAR file (or somewhere else on the classpath), we will keep them in the `etc/cas/google` directory in the overlay, which will result in their getting copied to `/etc/cas/google` when the server is deployed. This helps to ensure that they are not accidentally deleted.

Configure Google Apps properties

Add the following settings to `etc/cas/config/cas.properties` in the `cas-overlay-template` directory:

```
cas.googleApps.privateKeyLocation:  file:/etc/cas/google/privatekey.der
cas.googleApps.publicKeyLocation:   file:/etc/cas/google/publickey.der
cas.googleApps.keyAlgorithm:        RSA
```

References

- CAS 5: Google Apps Integration
- CAS 5: Configuration Properties: Google Apps
- Google: Generate Keys and Certificates for SSO

Configure Google single sign-on

Once the CAS side of things has been set up, the Google side has to be configured.

Configure SSO URLs

To configure a Google Apps domain to use the CAS server for user authentication:

1. Log in to the Google Admin Console for the domain to be configured.
2. Go to **Security > Set up single sign-on (SSO)**.
3. Check the **Setup SSO with third party identity provider** box
4. Enter the CAS login URL (`https://casdev.newschool.edu/cas/login`) in the **Sign-in page URL** blank. This should be the URL of the CAS login endpoing, ***not*** the URL of the CAS SAML2 IdP endpoint.
5. Enter the CAS logout URL (`https://casdev.newschool.edu/cas/logout`) in the **Sign-out page URL** blank.
6. Enter the URL a user should be directed to when changing his or her password in the **Change password URL** blank. This may or may not be a CAS endpoint, depending on whether the CAS password management feature has been configured.

> ⚠ **Important:** All URLs must be entered, and they must all use HTTPS.

Upload verification certificate

The X.509 file created earlier (page 394) has to be uploaded so that Google can verify sign-in requests.

After configuring the URLs and uploading the certificate, click the **SAVE** button.

References

- Google: Service provider SSO set up
- Google: SAML key and verification certificate

Install and test the application

Google Apps single sign-on can be tested by installing the updated CAS software on the CAS servers and restarting it, and then signing into Google.

Install and test on the master build server

Use the updated build and installation scripts (page 248) (or repeat the commands) to install the updated CAS server on the master build server (***casdev-master***) and restart Tomcat:

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# sh /opt/scripts/cassrv-install.sh
---Installing on casdev-master.newschool.edu
Installation complete.
casdev-master#
```

Review the contents of the log files (`/var/log/tomcat/catalina.yyyy-mm-dd.out` and `/var/log/cas/cas.log`) for errors.

Install on the CAS servers

Once everything is running correctly on the master build server, it can be copied to the CAS servers using the updated build and installation scripts (page 248):

```
casdev-master# sh /opt/scripts/cassrv-tarball.sh
casdev-master# for host in srv01 srv02 srv03
> do
> scp -p /tmp/cassrv-files.tgz casdev-${host}:/tmp/cassrv-files.tgz
> scp -p /opt/scripts/cassrv-install.sh casdev-${host}:/tmp/cassrv-in
stall.sh
> ssh casdev-${host} sh /tmp/cassrv-install.sh
> done
casdev-master#
```

Register Google Apps in the service registry

Using the management webapp, add a new service registry entry for the Google Apps instance. The service URL should be provided as:

```
^https://www.google.com/a/[Google Apps domain]/acs(\z|/.*)
```

For The New School then, this would be:

```
^https://www.google.com/a/newschool.edu/acs(\z|/.*)
```

> ⚠ **Important:** Although it uses the SAML2 protocol, Google Apps should be registered as a ***CAS client*** service, not a SAML2 Service Provider. That is, it should be registered as a `org.apereo.cas.services.RegexRegisteredService`, not as a `org.apereo.cas.support.saml.services.SamlRegisteredService`.

If the Google Apps instance does not use the same usernames as the CAS server, the service registry entry can be configured to return the alternate username on the *Username Attribute* tab of the management webapp.

Test the operation of Google Apps

To test the operation of Google Apps, visit `https://mail.google.com/a/[Google Apps domain]` (for The New School, `https://mail.google.com/a/newschool.edu`).

References

- CAS 5: Google Apps Integration

Moving to production

Summary: Moving from development and testing to production requires some additional steps to make sure everything goes smoothly.

The New School CAS 5 environment entered production over the University's 2018 Spring Break week.

The environment is essentially the one described in the introduction (page 9), with a total of five virtual servers (two in one data center, three in the other) operating in a pool behind a pair of F5 load balancers (one in each data center, in an active/passive configuration). Each virtual server is running a Tomcat instance (running both the CAS server and the CAS management webapp) and a MongoDB instance. The MongoDB instances are all members of the same replica set (which is why there are five servers; replica sets require an odd number of members) and handle the distributed ticket registry and distributed service registry.

The servers manage access to approximately 50 applications, hosted both locally in our data centers and remotely in the cloud by various Software-as-a-Service providers. Half a dozen of these applications are SAML2-based and authenticate through the CAS SAML2 IdP; the rest are CAS-based. Most of the applications require only the user principal name (username) or a single user attribute, although a few require more.

Event counts for Oct. 1 - Oct. 15, 2018

Event	Average Events/Day
Authentication Event Triggered	67,183
Authentication Success	21,905
Authentication Failed	2,126
Service Ticket Created	32,612
Service Ticket Not Created	15
Service Ticket Validated	22,224
Service Ticket Validate Failed	509
Ticket Granting Ticket Created	21,457

Event	Average Events/Day
Ticket Granting Ticket Destroyed	10,109

This chapter describes steps and configuration changes that should be considered when moving from a development and/or test environment to a production environment. It also describes some of the problems we have encountered after going live, and how we corrected them.

Configuration changes

Moving the CAS server from the development environment to the test environment, and then from the test environment to production, requires a number of changes to the configuration settings to identify the servers in the new environment, update encryption keys, etc. There are also some other configuration changes that should be made (or at least considered), especially when moving to the production environment.

Update CAS configuration settings

Several configuration settings created during the course of building the servers and enabling various features are specific to the environment in which the servers were built, and need to be updated when moving to another environment:

1. Several properties in `etc/cas/config/cas.proprties` must be updated with values for the new environment (in the list below, moving to the production environment is assumed):
    a. The `cas.server.name` property must be updated with the production server's domain name.
    b. If the production Active Directory/LDAP servers are not the same ones as those used by the development/test environment, the `cas.authn.ldap[n].ldapUrl` and `cas.authn.attributeRepository/ldapUrl` properties must be updated. The `bindDn` and `bindCredential` properties may also need to be updated.
    c. If the production environment uses different load balancers than the development/test environment, the `cas.adminPageSecurity.ip` regular expression must be updated to match the IP addresses of the production load balancers.
    d. The `mongo.hosts` "pseudo-property" from which the MongoDB connection string is constructed must be updated to contain the list of host names of the production MongoDB replica set members.
    e. New ticket granting cookie encryption keys (`cas.tgc.crypto.signing.key` and `cas.tgc.crypto.encryption.key`) should be generated for the production environment.
    f. New Spring Webflow encryption keys (`cas.webflow.crypto.signing.key` and

`cas.webflow.crypto.encryption.key`) should be generated for the production environment.

g. Thymeleaf caching should be enabled in the production environment by adding the `spring.thymeleaf.cache` property and setting its value to `true`.

2. Some properties in `etc/cas/config/management.properties` must also be updated:

   a. The `cas.server.name` property must be updated with the production server's domain name (***sso.newschool.edu***).

   b. The `mongo.hosts` "pseudo-property" from which the MongoDB connection string is constructed must be updated to contain the list of host names of the production MongoDB replica set members.

3. The CAS server SSL certificate in Tomcat's keystore must be replaced (page 42) with the production SSL certificate.

4. New SAML IdP metadata, encryption keys, and signing keys should be generated (page 195) for the production environment and copied to `etc/cas/saml`.

5. Additional CAS administrative users (who will have access to the management webapp and/or the dashboard) should be added to the `etc/cas/config/admusers.properties` file.

6. A new Duo protected application should be created (page 167) (via the Duo administration console) for the production CAS environment and the `duoIntegrationKey`, `duoSecretKey`, and `duoApplicationKey` components of the `cas.authn.mfa.duo[n]` property in `etc/cas/config/cas.properties` updated with the new values.

7. If the production Google Apps environment is not the same as the one used by the development/test environments, the key files in `etc/cas/google` should be regenerated (page 394) and the new X.509 certificate uploaded to the production Google Apps environment.

Once all of the above changes have been made in the `cas-overlay-template` directory on the master build server for the environment, the build and installation scripts (page 248) should be used to install and test the updated CAS server and management webapp on the master build server, and then everything should be copied to the CAS servers.

Lock down the Tomcat ROOT application

When we initially hardened the Tomcat installation (page 39), we chose not to remove the `ROOT` web application from Tomcat's `webapps` directory because it can be useful in a development/test environment to quickly determine whether Tomcat is working properly. We even [modified the application][setup_tomcat-test-the-tomcat-installation] to display the host name, IP address, and port number of the Tomcat server. This information can still be useful in a production environment for checking on load balancer issues and the like, but from a security perspective, it's not information that should be given to the general public.

To alleviate this problem, we can add some code to the top of the `/opt/tomcat/latest/webapps/ROOT/index.jsp` (`/var/lib/tomcat/ROOT/index.jsp`) file around line 18, just below this line:

```
<%@ page session="false" pageEncoding="UTF-8" contentType="text/html; charset=UTF-8" %>
```

This is the code to be added:

```
<%
  String a = request.getRemoteAddr();

  if (a != null && !a.substring(0, 10).equals("192.68.50.") &&
      !a.equals("192.168.1.10") && !a.equals("192.168.1.20")) {
    // response.sendRedirect("https://casdev.newschool.edu/cas/login");
    // response.sendRedirect("https://castest.newschool.edu/cas/login");
    response.sendRedirect("https://sso.newschool.edu/cas/login");
  }
%>
```

The code checks the IP address of the client host accessing the page and, if it's not the IT department subnet or the internal interface of one of the load balancers, sends an HTTP redirect to the CAS login page. With this addition, any attempt to access `https://sso.newschool.edu` from a non-whitelisted host will be redirected to `https://casdev.newschool.edu/cas/login`.

Note that the URL in the `response.sendRedirect()` call is dependent on the environment in which this code is running.

Set a default application

Occasionally, users will inadvertently create a browser bookmark for the CAS login page when what they (usually) mean to do is create a bookmark for the service that directed them there. Eventually, this results in the user visiting the CAS login page without having been directed there by a service. When this happens, and the user successfully authenticates, he or she will end up at a generic "login successful" page rather than at the service he or she actually wanted. Fortunately, CAS allows a different destination to be configured by adding a line to `etc/cas/config/cas.properties`:

```
cas.view.defaultRedirectUrl=https://my.newschool.edu
```

We have chosen to send users to *MyNewSchool*, the New School web portal—partly because it's a commonly-used application that is probably (arguably?) where they meant to go anyway, but more importantly, because it's an application where (almost) all users have an account. Sending a user to an application where he or she doesn't have an account is even more confusing that sending him or her to the generic login success page.

Create a standard location in which to store SAML SP metadata

When we created the service registry definition for our SAML2 test client (page 213), we provided a URL for the `metadataLocation` attribute, telling the CAS server that it could contact the SAML2 SP at that URL to dynamically download the SP's metadata. This is well and good for those SPs that support it, but at least in our experience many, if not most, SAML2-based SaaS applications do not offer dynamically-downloadable metadata. Instead, customers are expected to manually download the metadata and store it somewhere locally for their SAML2 IdP (CAS) to use.

Unfortunately, CAS does not yet support storing SAML SP metadata in a high-availability location such as the database where the ticket registry and service registry are stored. Therefore, ensuring that SAML SP metadata information is replicated across all the CAS servers in the pool must be handled manually. To make this at least a bit easier, we have chosen to maintain "master" copies of SP metadata files in the CAS Maven overlay, so that they can be distributed alongside any other updates to the CAS application or configuration files.

We have created an `etc/cas/saml/sp-metadata` subdirectory in the `cas-overlay-template` directory, and we store all third-party metadata files there. Whenever the CAS application or configuration files are updated, this subdirectory will be included in the files distributed by the build and installation scripts (page 248) we use for that purpose and extracted into `/etc/cas/saml/sp-metadata/` on each of the CAS servers in the pool.

> ℹ **Note:** While this method has worked reasonably well for us, it's by no means perfect. In practice, when a new SAML2 service is added, the system administrator responsible for doing that just places the SP metadata into `/etc/cas/saml/sp-metadata/[servicename].xml` on one of the CAS servers and manually copies it to the others with `scp`. Once everything is working, someone is supposed to copy the final `[servicename].xml` file back to the Maven overlay template so that it can be distributed properly in the future. We've forgotten this step once or twice though, so now the checklist for updating the Maven overlay (e.g., when a new version of CAS comes out) includes an item to check the servers for any new metadata files and copy them down to the overlay.

Logging to Graylog

Having multiple CAS servers behind a load balancer makes tracing things through log files difficult. Even with "sticky" sessions on the load balancer, there's no way to determine in advance which server in the pool is the one to be monitoring when trying to debug something. And if the servers are busy, going back and trying to find the one thing you needed after the fact can be difficult, as well. Since we have a Graylog server in our environment that lots of other things log to, it made sense to use that to consolidate CAS logs there as well.

> ☑ **Tip:** Syslog, of course, is another option. But log messages in Graylog Extended Log Format (GELF) provide significantly more information, in an easier-to-parse format, than log messages in typical Syslog format. This makes searching and filtering log messages much more robust on a Graylog server than on a Syslog server.

Configure Graylog to accept input from CAS

To configure Graylog to accept input from CAS:

1. Log into the Graylog server GUI and select **System > Inputs**.
2. From the "Select input" drop-down, choose `GELF UDP` and click the green **Launch new input** button.

3. Fill in some configuration settings:
    a. Under **Node**, choose the Graylog node that should host this input.
    b. Under **Title**, enter something meaningful like “Production CAS servers”.
    c. Under **Port**, choose a UDP port number to listen on (the default may be okay, but we chose to use different ports for the development, test, and production pools of CAS servers).
    d. Leave all the other fields at their defaults.
4. Click the maroon **Save** button.

Configure CAS to log to Graylog

To configure CAS to log to Graylog, edit the file `etc/cas/config/log4j2.xml` in the `cas-overlay-template` directory. First, define a socket appender by adding the following definition underneath all the `<RollingFile` appender definitions and just above the `<CasAppender>` definitions (around line 45):

```
<Socket name="graylog" host="graylog.newschool.edu" protocol="udp" po
rt="12203">
    <GelfLayout compressionType="GZIP" compressionThreshold="1024">
        <KeyValuePair key="webappName" value="cas"/>
    </GelfLayout>
</Socket>
```

Make sure that the `host` attribute contains the host name (or IP address) of the Graylog server and that the `port` attribute contains the value used when the new Graylog input was defined in the previous section.

> ❗ **Warning:** Graylog will accept input via TCP or UDP. However, if CAS is logging to Graylog via TCP, and the Graylog server becomes unavailable for some reason, the CAS servers can potentially run out of resources because all available threads are blocked waiting to deliver log messages to Graylog. We learned this the hard way—log to Graylog (or Syslog) with UDP.

Next, in the `<AsyncRoot>` definition (around line 108), add a reference to the Graylog appender (the `ref` attribute should have the same value as the `name` attribute in the `<Socket>` definition above):

```
<AsyncRoot level="warn">
    <AppenderRef ref="casFile"/>
    <AppenderRef ref="graylog"/>
    <!--
         For deployment to an application server running as service,
         delete the casConsole appender below
    -->
    <!-- <AppenderRef ref="casConsole"/> -->
</AsyncRoot>
```

In our experience, the `<AsyncRoot>` logger (`cas.log`) is the only one that needs to be sent to Graylog. Everything that gets logged to the audit logger (`cas_audit.log`) is also logged to the root logger, and the `perfStatsLogger` (`perfStats.log`) is only logging performance statistics. On the Tomcat side, `catalina.out` is only helpful if the CAS application itself is failing, and if that's happening in production, it's probably isolated to a single server.

Install the modified Log4j2 configuration

Install the updated `log4j2.xml` file on all the CAS servers:

```
casprod-master# cd /opt/workspace/cas-overlay-template
casprod-master# cat etc/cas/config/log4j2.xml > /etc/cas/config/log4j
2.xml
casprod-master# cd /
casprod-master# for h in 01 02 03 04 05
> do
> tar cf - etc/cas/config/log4j2.xml | ssh casprod-srv${h} "cd /; ta
r xf -"
> done
casprod-master#
```

CAS monitors `/etc/cas/config/log4j2.xml` for changes and picks them up automatically, so there is no need to restart the application.

Configure the management webapp to log to Graylog

Since the management webapp also runs on every server in the pool, it makes sense to send its logs to Graylog as well (it can log to the same Graylog input as the associated CAS servers). Repeat the steps above to add the Graylog `<Socket>` definition and the Graylog `<AppenderRef` > to `etc/cas/config/log4j2-management.xml` in the `cas-management-overlay` directory, and then install the modified configuration file on all the CAS servers.

Use an Extended Validation (EV) SSL certificate

To help users identify phishing scams that make use of a forged New School SSO login page, we purchased an Extendned Validation (EV) SSL certificate for our production CAS servers, which use the ***sso.newschool.edu*** host name. Because all the most popular browsers (except Safari) display the organization name associated with the certificate in the URL bar when an EV certificate is used, it's easy to instruct users to look for this to confirm that they're accessing the "official" login page.

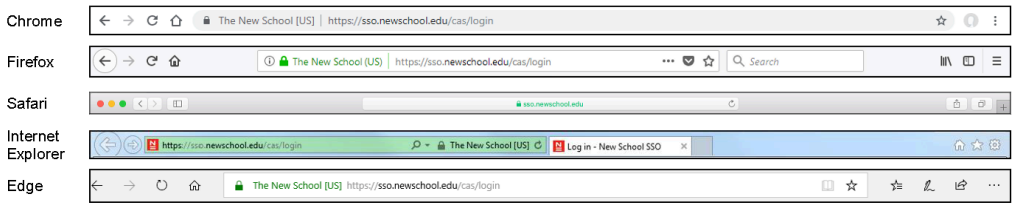

Figure 27. How browsers display EV SSL certificates

Problems encountered

For the most part, CAS 5 in production has worked quite well. However, as we moved through the end of the Spring 2018 semester, through the summer, and then the start of the Fall semester, we did encounter a few problems that had to be addressed.

Incorrect MongoDB connection pool size

At the beginning of the fall semester, we began to receive support desk calls from users who had successfully authenticated to CAS, but upon being redirected back to the calling service, were receiving an error that the service could not locate the service ticket. We had seen this sporadically before, and simply refreshing the browser was usually enough to cure the problem, but it was becoming a larger problem. We pretty quickly determined that the problem had something to do with tickets being requested from the MongoDB ticket registry before they had been written, but figuring out what was causing that to happen was much more difficult.

Eventually, we found the problem. It turns out that the CAS code that makes calls to the MongoDB Java driver has not kept up with driver developments on the MongoDB side; it is configuring the driver with deprecated property settings. This ultimately results in the driver being configured with a connection pool that is an order of magnitude too small—50 connections instead of 500 (the default, if non-deprecated property settings are used). The increased load on our servers as a result of the fall semester's start, combined with virtual server sizes that were barely adequate (see below) was just enough to cause the CAS servers to occasionally deplete the connection pool.

The solution to the problem was to add parameters to the MongoDB connection string to explicitly set the size of the connection pool back to the defaults. This was done by changing the value of the `mongo.opts` "pseudo-property" in `cas.properties` and `cas-management.properties` from

```
mongo.opts:                                    &ssl=true
```

to

```
mongo.opts:                                    &ssl=true&maxPoolSize=100&wai
tQueueMultiple=5
```

Since making this change (and increasing the size of the servers, see below) we have not seen any more occurrences of this problem.

Limit MongoDB cache size

As one step in investigating the problem described above, we took a look at the MongoDB internal cache. By default, MongoDB will set the size of this cache to the larger of

- 50% of (RAM - 1GB), or
- 256MB

On our 8GB servers, this results in a cache size of 3.5GB ((8GB - 1GB) × 0.5). As it turns out, for our environment, this size performs pretty well, and it didn't make any sense to us to increase it. However, because we had separately made the decision to increase the CAS servers to 12GB of memory (see below), we realized that we needed a way to limit the size of MongoDB's cache, or it would start using 5.5GB ((12GB - 1GB) × 0.5) on the new servers.

To limit the size of MongoDB's cache, edit the `/etc/mongodb.conf` configuration file on the master build server for the environment and locate the `storage` section (around line 13) and change it from:

```
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true
#  engine:
#  mmapv1:
#  wiredTiger:
```

to

```
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true
  wiredTiger:
    engineConfig:
      cacheSizeGB: 4
#  engine:
#  mmapv1:
```

This will limit the cache size to 4GB regardless of how much memory is on the system. Copy `/etc/mongod.conf` to all of the CAS servers in the environment and restart MongoDB.

Server sizes in production

When we first went into production, the virtual servers were configured with 2 CPUs and 8GB of memory. This worked well until we entered the start of the Fall semester (one of our heaviest load periods), when we began to notice the servers slowing down because they were starved for resources. In part this was caused by the two MongoDB configuration issues described above, but it was also because things were running right on the edge already, and the additional load from the start of the semester was enough to push them over.

We have since decided to increase the size of the virtual servers to 4 CPUs and 12GB of memory. We have kept the Java heap size limited to 4GB (which is plenty), and have now also limited the MongoDB cache size to 4GB (see above). This results in two-thirds of the available memory on the system being devoted to CAS, and the remaining one-third being left for the kernel, other processes, etc.

Since making these changes, the servers have been performing very well.

Cleaning up log files

In our environment, we don't want log files to grow without bound, nor do we want to accumulate log files "forever." In fact, we have configured CAS to rotate its log file(s) every day (page 97), and we really don't need to keep more than the last 30 days' worth of logs on line (we can go back further via backups, if necessary, plus we have everything in Graylog). Unfortunately, convincing Log4j2 to do this appears to be the next best thing to impossible, and getting Tomcat's JULI-based logging to do it is even worse. After trying numerous configurations without any luck, we eventually gave up and created `/etc/cron.daily/caslogs`:

```
#!/bin/sh
#
# Clean up CAS log files in /var/log/tomcat and /var/log/cas. These f
iles do
# not necessarily get created every day, so deleting by age might en
d up
# deleting the current file. Instead, we keep the last $NLOGS files o
f
# each type.
#

PATH=/bin:/usr/bin; export PATH

TMP=/tmp/caslog$$
NLOGS=30

trap "rm -f $TMP" 0

DATEGLOB='[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]'

if [ -d /var/log/tomcat ]
then
    cd /var/log/tomcat

    for f in catalina host-manager localhost localhost_access_log man
ager
    do
        ls -1 ${f}.${DATEGLOB}.* | head -n -${NLOGS} > ${TMP}

        if [ -s ${TMP} ]
        then
            rm -f `cat ${TMP}`
        fi
    done
fi

if [ -d /var/log/cas ]
then
    cd /var/log/cas

    for f in cas cas-management cas_audit perfStats
    do
        ls -1 ${f}-${DATEGLOB}.log | head -n -${NLOGS} > ${TMP}

        if [ -s ${TMP} ]
        then
            rm -f `cat ${TMP}`
```

```
        fi
    done
fi

exit 0
```

This script is executed every night at (approximately) midnight by `cron`. It will keep the `NLOGS` most recent log files of each type in `/var/log/cas` and `/var/log/tomcat`, and delete any beyond that number. It relies on all the log files having names of the general format

```
[basename].YYYY-MM-DD.*
```

It does not make any assumptions about how often the log file is rotated; it will always keep the most recent `NLOGS` files regardless of how old they are. Although we have not seen the need to yet, the script could easily be modified to compress the remaining log files with `gzip` or something else.

Fixing a bug in Duo's WebSDK

Once we started rolling out Duo MFA, we discovered that some users—those who were using Internet Explorer—were having difficulties using it because the "Duo box" was not appearing after they entered their username and password. Instead, a blank page would appear and the login process would essentially be "stuck" at that point. To make things even more interesting, the problem only seemed to occur when the users were accessing a SAML2-based service (SP); services that authenticated via CAS and required Duo MFA did not exhibit the problem. Eventually, we determined that the problem was due to a bug in the Duo Web SDK (written by Duo Security) used by the CAS server.

How to fix the problem

Fixing the problem requires making a change to `Duo-Web-v2.js` and then installing the corrected file in the CAS server. The change to be made is:

```
*** Duo-Web-v2.js       2018-06-28 08:12:08.723891501 -0400
--- Duo-Web-v2-fix.js   2018-06-28 08:14:41.721450104 -0400
***************
*** 374,380 ****
          // point the iframe at Duo
          iframe.src = [
              'https://', host, '/frame/web/v1/auth?tx=', duoSig,
!             '&parent=', encodeURIComponent(document.location.href),
              '&v=2.6'
          ].join('');

--- 374,383 ----
          // point the iframe at Duo
          iframe.src = [
              'https://', host, '/frame/web/v1/auth?tx=', duoSig,
!             '&parent=',
!           (window.postMessage ?
!               encodeURIComponent(document.location.href.spli
t('?')[0]) :
!               encodeURIComponent(document.location.href)),
              '&v=2.6'
          ].join('');
```

To apply the patch and include it in the CAS server:

1. Download the original source from GitHub .
2. Either make the change above manually, or save it to a file and run `patch -p0 < patch.txt` .
3. Feed the patched file to your favorite JavaScript minimizer (e.g., Uglify) and save the result.
4. Copy the new minimized file to the Maven overlay at `src/main/resources/static/js/duo/Duo-Web-v2.js` or, if you have configured your own theme, at `src/main/resources/static/themes/yourtheme/js/duo/Duo-Web-v2.js` , and rebuild and install the application.

Explanation of the patch

The problem is that as the web flow goes back and forth between CAS and the SP, the query parameters that CAS puts on the end of the URL get longer and longer. This seems to happen with all CAS/SAML2 services, but it's much worse when one or both sides of the SAML2 negotiation require signed and/or encrypted assertions; the URL with query parameters can grow in length to tens of thousands of characters.

Eventually things get to the point in the webflow where CAS sends `casDuoLoginView.html` to the user's browser. This is the page that includes the Duo Web SDK, a bunch of JavaScript that manipulates the `iframe` on the page that displays the Duo dialog. To populate the Duo `iframe`, the Web SDK constructs a source URL that points at the Duo back-end servers and includes as a query parameter the full URL of the CAS server including all of its query parameters. This is where the problem appears—because that URL is already really, really long, the resulting source URL calling back to the Duo back-end is even longer, and it exceeds the maximum length of a URL in Internet Explorer (2,083 characters) causing IE11 to silently fail, and so you end up with a blank `iframe`. (In our testing, Edge was also impacted, even though according to its documentation it should not have been.)

We reported this to Duo back in May 2018 and provided a simple one-line fix, which was just to truncate the query parameters off the CAS URL before sending it to the Duo back-end. The response back from the Duo engineer was that by doing that we "may experience breakage with any users using IE7 or other browsers that don't have `window.postMessage` natively supported."

So they're apparently, for whatever reason, trying to maintain backward compatibility with some pretty ancient browsers—`window.postMessage` has been supported by all major browsers since 2009. But okay, to help preserve their backward compatibility we suggested a slightly "smarter" version of the patch to the Duo engineer—one that only truncates the URL if `window.postMessage` is supported by the browser (in which case it doesn't need the URL anyway). That's what the patch above does. But we never got a response back to that suggestion, so we ended up making the patch ourselves.

About The New School

Our history

In 1919, a few great minds imagined a school that would rethink the purpose of higher learning. The New School is an amalgamation of a world-famous design school, a premier liberal arts college, a renowned performing arts school, a legendary social research school, and other advanced degree programs, created to challenge and support freethinkers who want to change the world.

No matter what your age or stage of life, you can find more to learn at The New School. With over 135 undergraduate and graduate degree programs, our university offers a more creatively inspired, rigorously relevant education than any other. Our university's urban academic centers in New York City, Paris, and Mumbai offer almost 10,000 students the chance to expand their knowledge and view of the world.

Our schools

No matter what area of study students pursue at The New School, they will discover a unique form of creative problem solving that will forever change the way they investigate and create. They will learn to relentlessly question convention, collaborate across disciplines, and take risks. They will immerse themselves in a world of critical analysis and intense scholarship. And ultimately, they will seek new ways to effect positive change. Five progressive colleges inspire world-changing creativity:

- Parsons School of Design
- Eugene Lang College of Liberal Arts
- College of Performing Arts
- The New School for Social Research
- Schools of Public Engagement

For more information, visit www.newschool.edu .

Author information

This documentation was created by:

David A. Curry, CISSP
Director of Information Security

The New School
71 Fifth Avenue, 9th Floor
New York, New York 10003
david.curry@newschool.edu

The author would like to thank Misagh Moayyed, Dmitriy Kopylenko, Jérôme Leleu, Travis Schmidt, and the members of the CAS Community for the help in answering questions and suggesting solutions during the development of this project.