Deploying Apereo CAS

(DRAFT)

Last generated: November 03, 2017

Table of Contents

Introduction

Setting up the environment

Install Apache Tomcat on the CAS servers

Install HTTPD and PHP on the client servers

Building the CAS server

Adding a service registry

Building the CAS client

Adding LDAP support

Configuring LDAP authentication

Configuring LDAP attribute resolution and release

Adding MFA support

Adding SAML support

Building the SAML client

Enabling the dashboard (adminpages)

Building the service management webapp

Customizing the CAS user interface

High availability

Building the configuration server

Setting up the service registry

Setting up the ticket registry

About

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.

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 services 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.

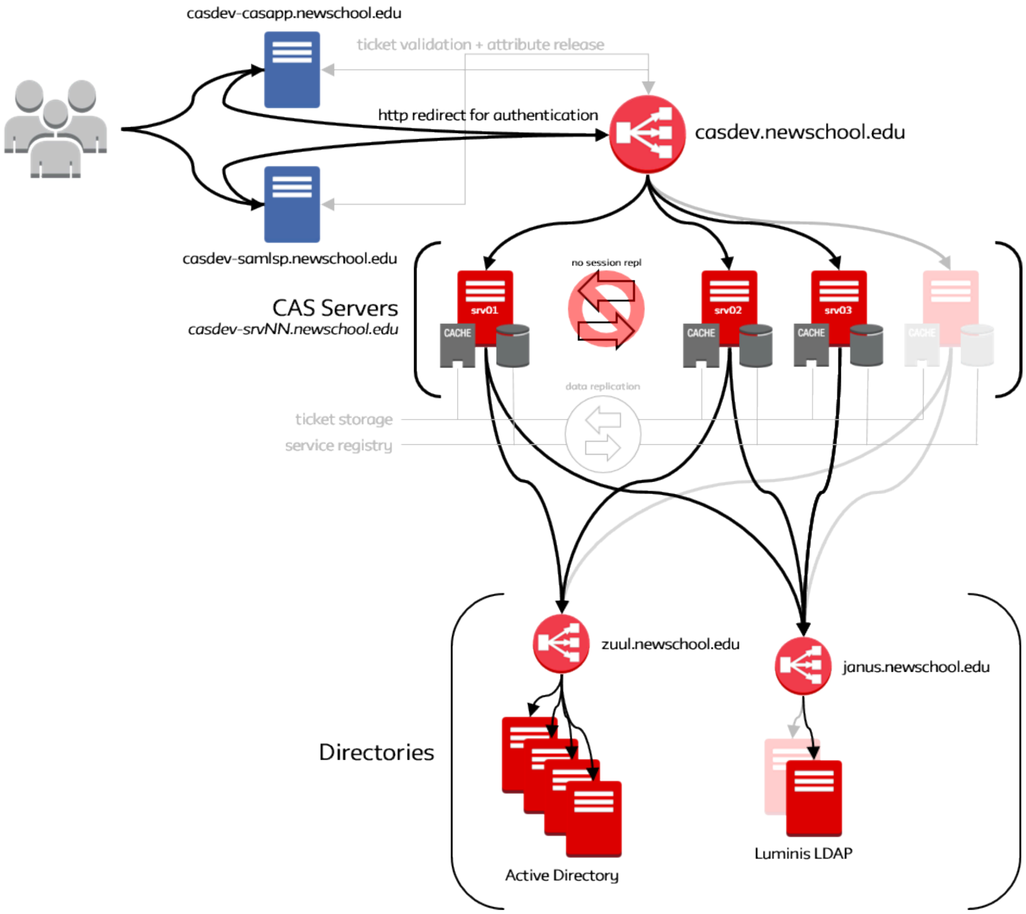


Figure 1. The New School CAS 5 development environment

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)

ℹ **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.

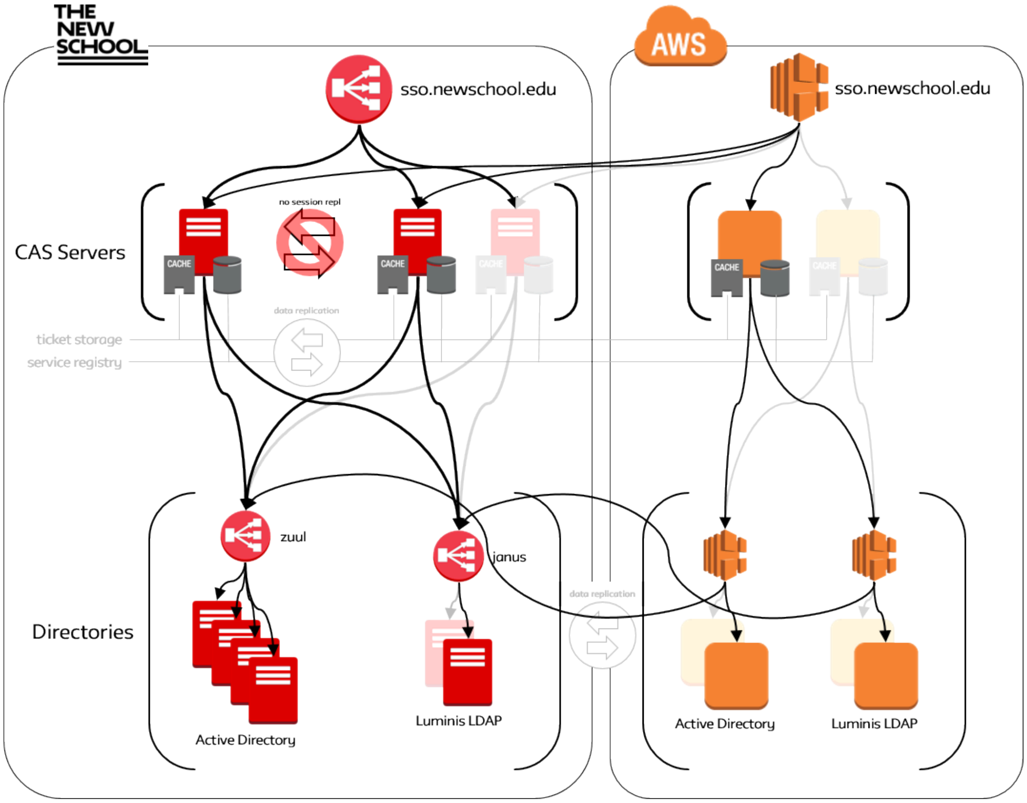


Figure 2. Hybrid on-premises/cloud CAS 5 environment

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 services 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 2 CPUs with 4 GB of RAM and 20 GB of disk space, which are the minimums recommended in the CAS 5 documentation.

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 85), 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 24)). 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.

ℹ **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

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 24), 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
8.5.20/bin/apache-tomcat-8.5.20.tar.gz'
  % Total    % Received % Xferd  Average Speed   Time    Time     Tim
e  Current
                                 Dload  Upload   Total   Spent    Lef
t  Speed
100 9212k  100 9212k    0     0  5191k      0  0:00:01  0:00:01
--:--:-- 5192k
casdev-master# tar xzf apache-tomcat-8.5.20.tar.gz
casdev-master# ln -s apache-tomcat-8.5.20 latest
casdev-master# rm -f apache-tomcat-8.5.20.tar.gz
```

to download and install Tomcat on the master build server (***casdev-master***). (Replace `8.5.20` 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 24), 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
f.tar.gz'
% Total    % Received % Xferd  Average Speed   Time    Time     Tim
e  Current
                               Dload  Upload   Total   Spent    Lef
t  Speed
100 5154k  100 5154k    0     0  7543k      0 --:--:-- --:--:--
--:--:-- 7546k
casdev-master# tar xzf openssl-1.1.0f.tar.gz
casdev-master# cd openssl-1.1.0f
casdev-master# mkdir -p /opt/openssl/openssl-1.1.0f
casdev-master# ln -s openssl-1.1.0f /opt/openssl/latest
casdev-master# ./config --prefix=/opt/openssl/openssl-1.1.0f 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-1.1.0f openssl-1.1.0f.tar.gz
```

to download and build OpenSSL on the master build server (***casdev-master***). (Replace `1.1.0f` 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-1.6.2.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-1.6.2.tar.gz
casdev-master# cd apr-1.6.2
casdev-master# mkdir -p /opt/apr/apr-1.6.2
casdev-master# ln -s apr-1.6.2 /opt/apr/latest
casdev-master# ./configure --prefix=/opt/apr/apr-1.6.2
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-1.6.2 apr-1.6.2.tar.gz
```

to download and build the APR library on the master build server (***casdev-master***). (Replace `1.6.2` 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-8.5.20/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-8.5.20
(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 `8.5.20` 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-8.5.20/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 `8.5.20` 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 38)). 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 tomcat
```

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 casdev.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 incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or 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.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-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 55)), 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.Http11NioProtocol"
            maxThreads="150" SSLEnabled="true">
     <SSLHostConfig>
         <Certificate certificateKeystoreFile="conf/localhost-rsa.jks"
                      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 43), above.

ℹ **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</servlet-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-class>
    <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 41)) 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 &quot;%r&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 45)) 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
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>Tomcat Secure HTTP (HTTPS)</short>
  <description>Tomcat typically implements TLS/SSL-secured HTTP (HTTPS) 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 -server'

# 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.ClassLoaderLogManager \
            -Djava.util.logging.config.file=${CATALINA_BASE}/conf/logging.properties \
            -cp ${CATALINA_HOME}/bin/commons-daemon.jar:${CATALINA_HOME}/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
```

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 41) 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
```

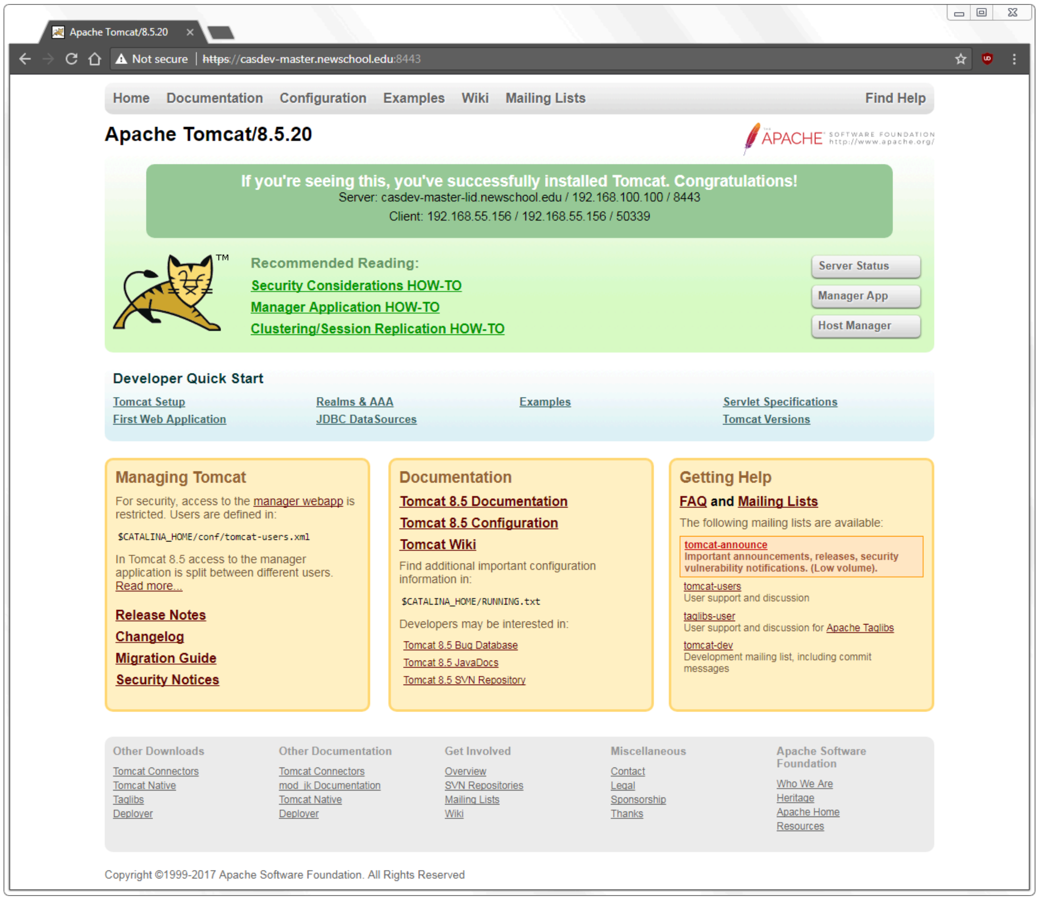
then the Native Library was not installed correctly, and the installation needs to be fixed.

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:



Apache Tomcat/8.5.20

If you're seeing this, you've successfully installed Tomcat. Congratulations!
Server: casdev-master-lid.newschool.edu / 192.168.100.100 / 8443
Client: 192.168.55.156 / 192.168.55.156 / 50339

Recommended Reading:
Security Considerations HOW-TO
Manager Application HOW-TO
Clustering/Session Replication HOW-TO

Developer Quick Start

Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat

For security, access to the manager webapp is restricted. Users are defined in:

```
$CATALINA_HOME/conf/tomcat-users.xml
```

In Tomcat 8.5 access to the manager application is split between different users. Read more...

Release Notes
Changelog
Migration Guide
Security Notices

Documentation

Tomcat 8.5 Documentation
Tomcat 8.5 Configuration
Tomcat Wiki

Find additional important configuration information in:

```
$CATALINA_HOME/RUNNING.txt
```

Developers may be interested in:

Tomcat 8.5 Bug Database
Tomcat 8.5 JavaDocs
Tomcat 8.5 SVN Repository

Getting Help

FAQ and Mailing Lists

The following mailing lists are available:

tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).

tomcat-users
User support and discussion

taglibs-user
User support and discussion for Apache Taglibs

tomcat-dev
Development mailing list, including commit messages

Other Downloads
Tomcat Connectors
Tomcat Native
Taglibs
Deployer

Other Documentation
Tomcat Connectors
mod_jk Documentation
Tomcat Native
Deployer

Get Involved
Overview
SVN Repositories
Mailing Lists
Wiki

Miscellaneous
Contact
Legal
Sponsorship
Thanks

Apache Software Foundation
Who We Are
Heritage
Apache Home
Resources

Copyright ©1999-2017 Apache Software Foundation. All Rights Reserved

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 41) 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/openssl opt/tomcat var/cache/tomcat var/lib/tomcat var/cache/tomcat etc/firewalld/services/tomcat-https.xml etc/systemd/system/tomcat.service
```

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.

> **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-install.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 103)).

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 41); 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 50), 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 73), 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

> ℹ **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-scale=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.</big></p>
        <?php phpinfo(); ?>
    </div>
  </body>
</html>
```

> ℹ **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
```

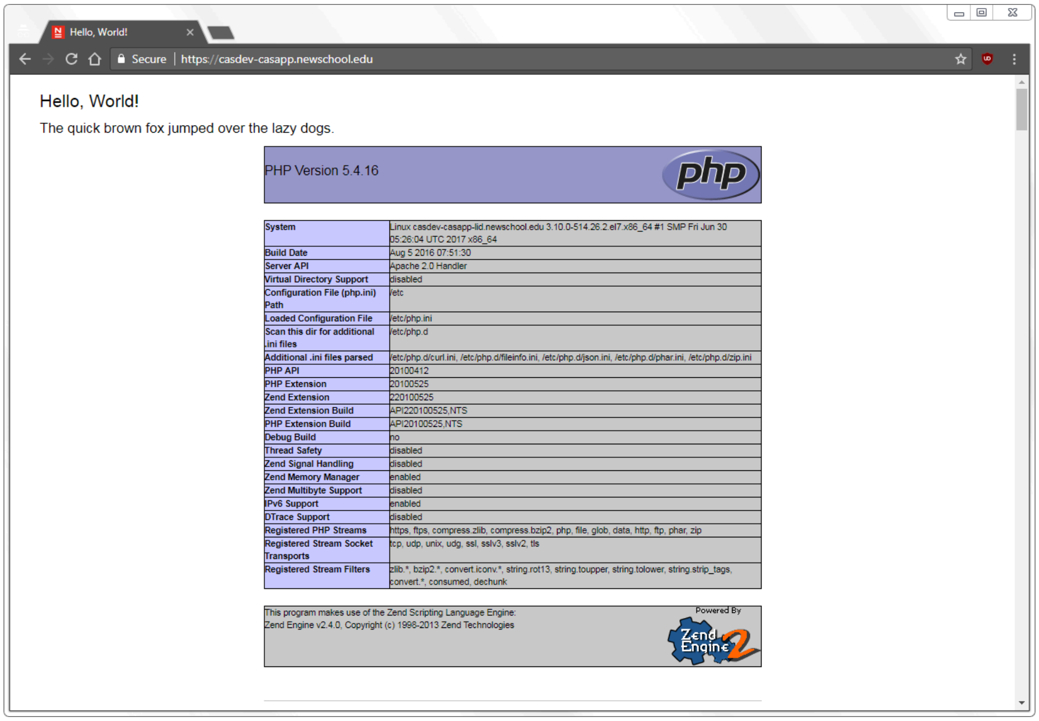
Review the contents of the log files in the `/var/log/httpd` directory for errors.

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:



Hello, World!

The quick brown fox jumped over the lazy dogs.

PHP Version 5.4.16

System	Linux casdev-casapp-lid.newschool.edu 3.10.0-514.26.2.el7.x86_64 #1 SMP Fri Jun 30 05:26:04 UTC 2017 x86_64
Build Date	Aug 5 2016 07:51:30
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/curl.ini, /etc/php.d/fileinfo.ini, /etc/php.d/json.ini, /etc/php.d/phar.ini, /etc/php.d/zip.ini
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension Build	API220100525,NTS
PHP Extension Build	API20100525,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.4.0, Copyright (c) 1998-2013 Zend Technologies

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 services 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-template.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 89)).

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_cnxEqYo9Bb384V70eZIvWctQ5V6xTO4P6wsQjFlglD9OSQNlFdb0mT2Q1E3qXdo05_tzrjQ
cas.tgc.crypto.encryption.key:        r88iOMdbRMLOkITV54kax4WgadTdzUYSBXNhOp_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 38) 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 the `size="10MB"` attribute from the `SizeBasedTriggeringPolicy`

element (around line 22).

3. 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="${sys:cas.log.dir}/cas.log" append="true"
             filePattern="${sys:cas.log.dir}/cas-%d{yyyy-MM-dd}.log">
    <PatternLayout pattern="%d %p [%c] - &lt;%m&gt;%n"/>
    <Policies>
        <OnStartupTriggeringPolicy />
        <SizeBasedTriggeringPolicy />
        <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).

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 38), 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=tomcat --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 --mode=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:

```sh
#!/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 58) 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:

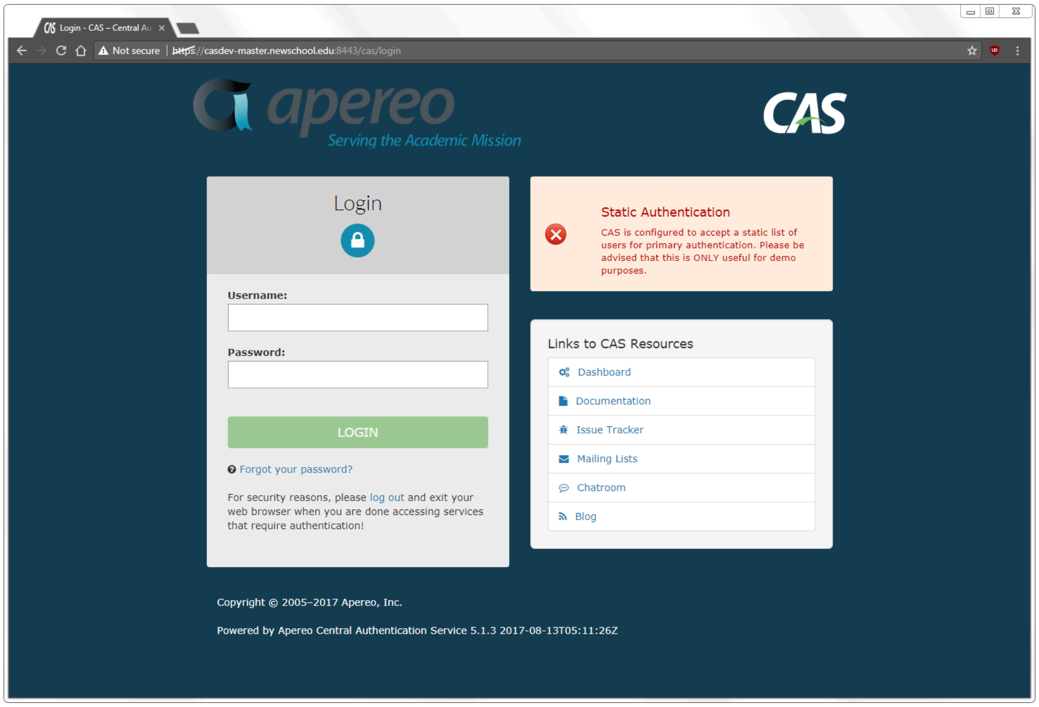
```
YYYY-MM-DD HH:MM:SS,sss WARN [org.apereo.cas.config.CasCoreTicketsCon
figuration] - <Runtime memory is used as the persistence storage for
retrieving and managing tickets. Tickets that are issued during runti
me will be LOST upon container restarts. This MAY impact SSO function
ality.>
YYYY-MM-DD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


Login - CAS – Central Au
Not secure
https://casdev-master.newschool.edu:8443/cas/login
apereo
Serving the Academic Mission
CAS
LOGIN

Login

Username:

Password:

Forgot your password?

For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!

Static Authentication

CAS is configured to accept a static list of users for primary authentication. Please be advised that this is ONLY useful for demo purposes.

Links to CAS Resources

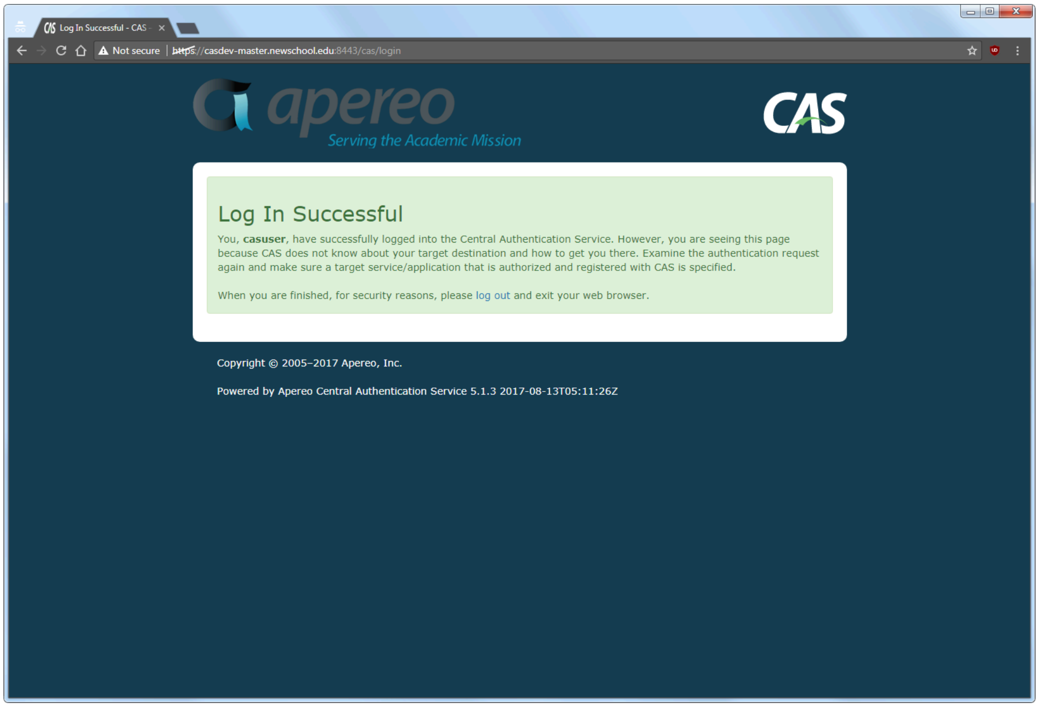
- Dashboard
- Documentation
- Issue Tracker
- Mailing Lists
- Chatroom
- Blog

Copyright © 2005–2017 Apereo, Inc.

Powered by Apereo Central Authentication Service 5.1.3 2017-08-13T05:11:26Z

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:


Log In Successful - CAS -
Not secure
https://casdev-master.newschool.edu:8443/cas/login
apereo
Serving the Academic Mission
CAS

Log In Successful

You, **casuser**, have successfully logged into the Central Authentication Service. However, you are seeing this page because CAS does not know about your target destination and how to get you there. Examine the authentication request again and make sure a target service/application that is authorized and registered with CAS is specified.

When you are finished, for security reasons, please log out and exit your web browser.

Copyright © 2005–2017 Apereo, Inc.

Powered by Apereo Central Authentication Service 5.1.3 2017-08-13T05:11:26Z

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 65), 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 236).

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 service management webapp (page 234) 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 98) (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#
```

ℹ **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 98):

```
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:

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello, World!</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=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.</big></p>
        <p><big>Click <a href="secured-by-cas/index.php">here</a> for 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-scale=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
```

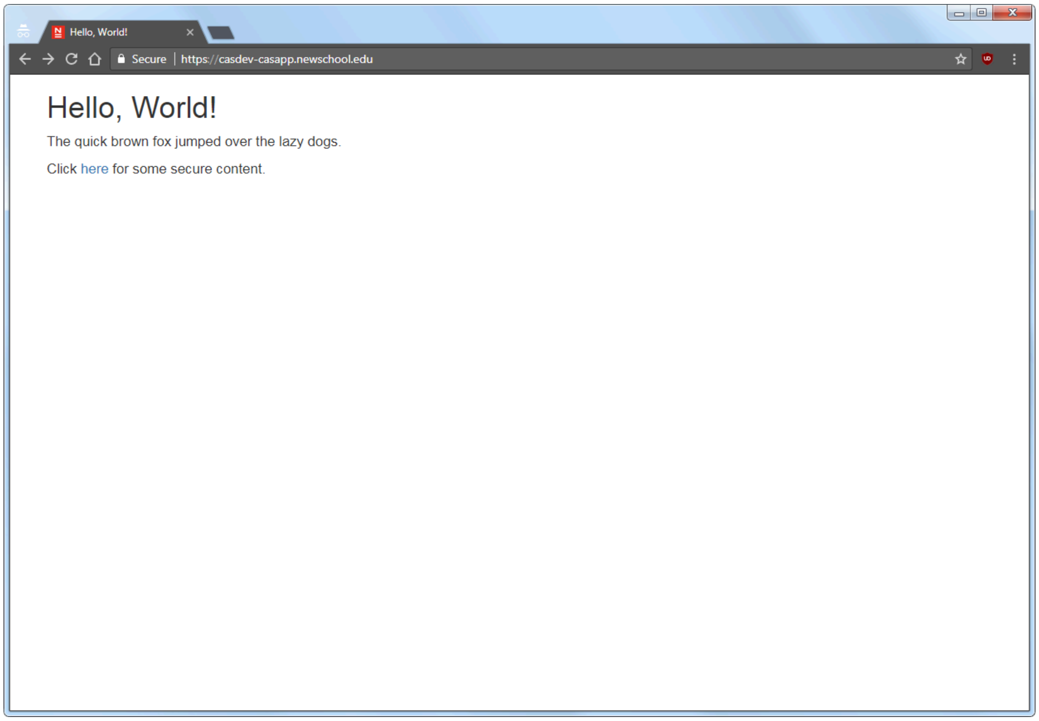
on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

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:



Hello, World!

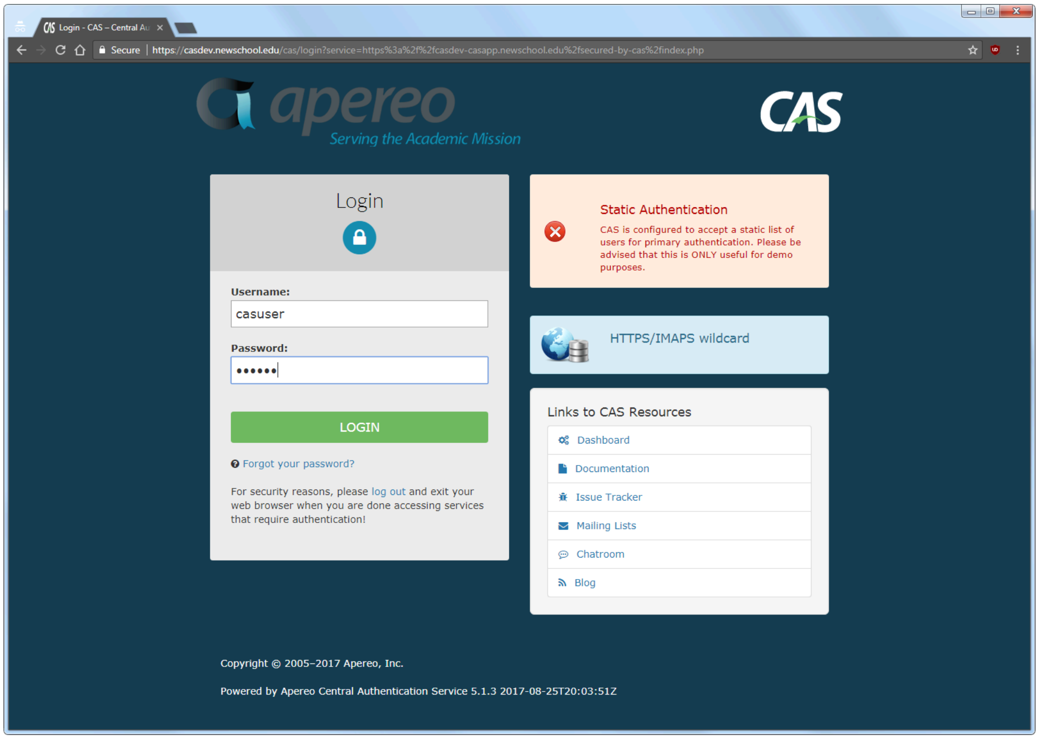
The quick brown fox jumped over the lazy dogs.

Click here for some secure content.

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:


Login - CAS – Central Au
Secure
https://casdev.newschool.edu/cas/login?service=https%3a%2f%2fcasdev-casapp.newschool.edu%2fsecured-by-cas%2findex.php
apereo
Serving the Academic Mission
CAS
LOGIN
Forgot your password?
Dashboard
Documentation
Issue Tracker
Mailing Lists
Chatroom
Blog

Login

Username: casuser

Password: ••••••

For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!

Static Authentication

CAS is configured to accept a static list of users for primary authentication. Please be advised that this is ONLY useful for demo purposes.

HTTPS/IMAPS wildcard

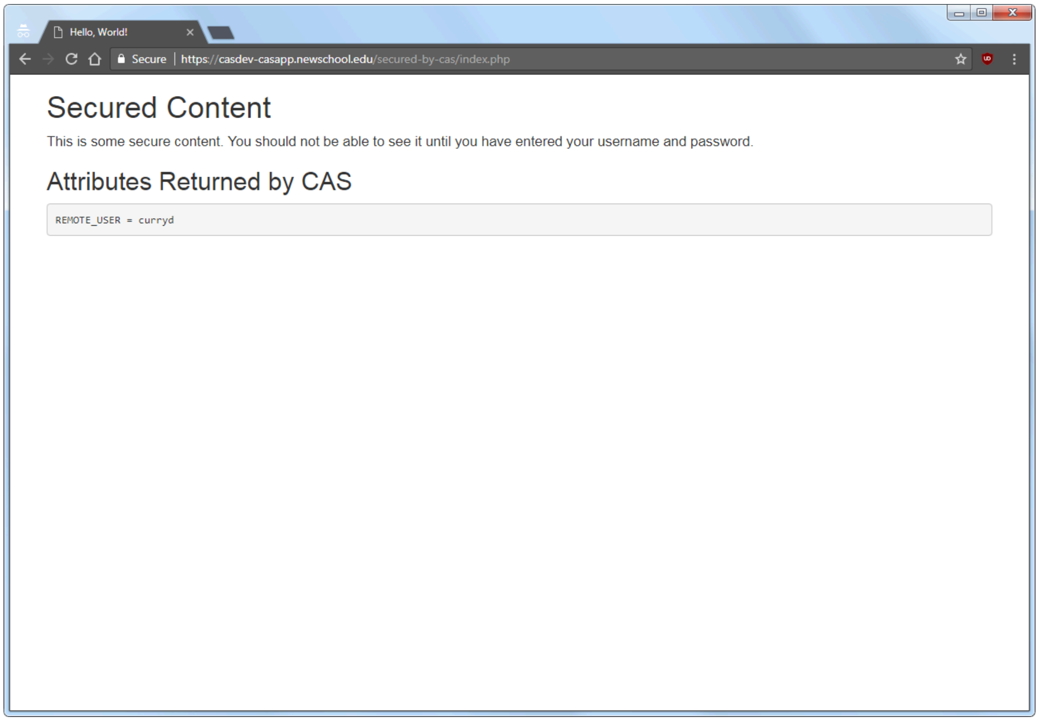
Links to CAS Resources

Copyright © 2005–2017 Apereo, Inc.

Powered by Apereo Central Authentication Service 5.1.3 2017-08-25T20:03:51Z

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:


Hello, World!
Secure
https://casdev-casapp.newschool.edu/secured-by-cas/index.php

Secured Content

This is some secure content. You should not be able to see it until you have entered your username and password.

Attributes Returned by CAS

```
REMOTE_USER = curryd
```

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</artifactId>
        <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</artifactId>
        <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].userFilter:           sAMAccountName={user}
cas.authn.ldap[0].baseDn:               ou=TNSUsers,dc=tns,dc=newschool,dc=edu
cas.authn.ldap[0].dnFormat:             cn=%s,ou=TNSUsers,dc=tns,dc=newschool,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 135)).

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.
`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 98) (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 98):

```
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].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
cas.authn.ldap[1].principalAtrributeId: uid
```

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.
`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.
`principalAttributeId`	The attribute to be used as the security principal identifier. In our case, `uid` is the username, which is what we want.

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.properties
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-${host}:/etc/cas/config/ca
s.properties
> ssh casdev-${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://zuul.newschool.edu
cas.authn.attributeRepository.ldap[0].userFilter:       sAMAccountName={user}
cas.authn.attributeRepository.ldap[0].baseDn:           ou=TNSUsers,dc=tns,dc=newschool,dc=edu
cas.authn.attributeRepository.ldap[0].bindDn:           cn=ldap_ssotest,ou=Service,ou=Users,ou=Enterprise Support,dc=tns,dc=newschool,dc=edu
cas.authn.attributeRepository.ldap[0].bindCredential:   xxxxxxxxxxxx
cas.authn.attributeRepository.ldap[0].attributes.cn:    uid
cas.authn.attributeRepository.ldap[0].attributes.displayName:   displayName
cas.authn.attributeRepository.ldap[0].attributes.givenName:     givenName
cas.authn.attributeRepository.ldap[0].attributes.mail:  mail
cas.authn.attributeRepository.ldap[0].attributes.sn:    sn
cas.authn.attributeRepository.ldap[0].attributes.tnsGoogleAppsRole:     role
cas.authn.attributeRepository.ldap[0].attributes.tnsIDNumber:   cn
```

The first six 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://janus.newschool.edu
cas.authn.attributeRepository.ldap[1].userFilter:       uid={user}
cas.authn.attributeRepository.ldap[1].baseDn:           ou=People,o=cp
cas.authn.attributeRepository.ldap[1].bindDn:           uid=ldap_ssotest,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:   displayName
cas.authn.attributeRepository.ldap[1].attributes.givenName:     givenName
cas.authn.attributeRepository.ldap[1].attributes.mail:  mail
cas.authn.attributeRepository.ldap[1].attributes.sn:    sn
cas.authn.attributeRepository.ldap[1].attributes.udcid: UDC_IDENTIFIER
cas.authn.attributeRepository.ldap[1].attributes.uid:   uid
```

As above, the first six 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 109), 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-cas(\\z|/.*)",
  "name" : "Apache Secured By CAS",
  "id" : 1504122840,
  "description" : "CAS development Apache mod_auth_cas server with username/password protection",
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePolicy"
  },
  "evaluationOrder" : 1100
}
```

Following the naming convention introduced earlier (page 109), 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 service management webapp (page 234).

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-mapped(\\z|/.*)",
  "name" : "Return Mapped Test",
  "id" : 1506518400,
  "description" : "Display results of a Return Mapped attribute release policy",
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnMappedAttributeReleasePolicy",
    "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-scale=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.</big></p>
        <p><big>Click <a href="secured-by-cas/index.php">here</a> for some
          content secured by username and password.</big></p>
        <p><big>Click <a href="return-mapped/index.php">here</a> to see the
          results of the "Return Mapped" attribute release policy.</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.

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 98) (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 98):

```
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-install.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
```

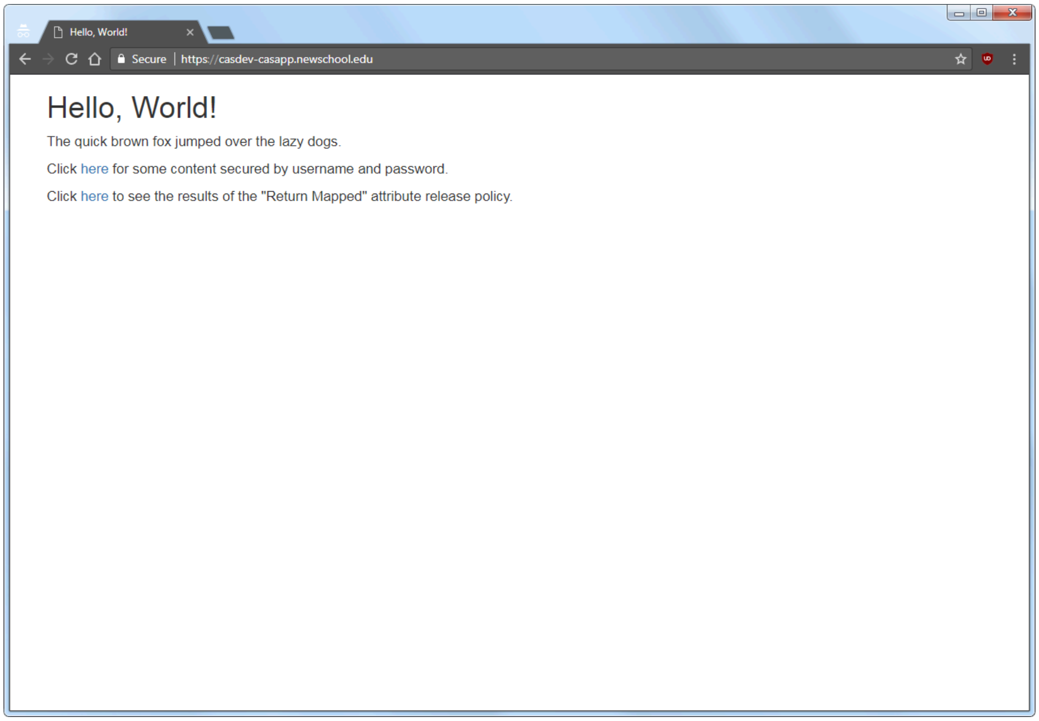
on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

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:



Hello, World!

The quick brown fox jumped over the lazy dogs.

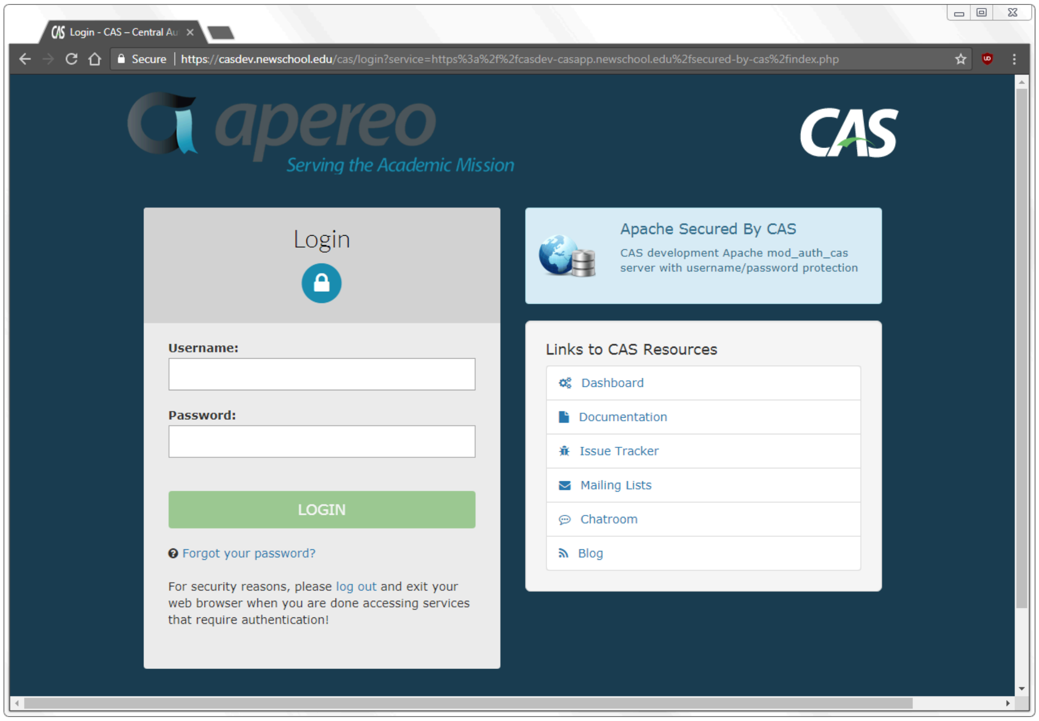
Click here for some content secured by username and password.

Click here to see the results of the "Return Mapped" attribute release policy.

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:

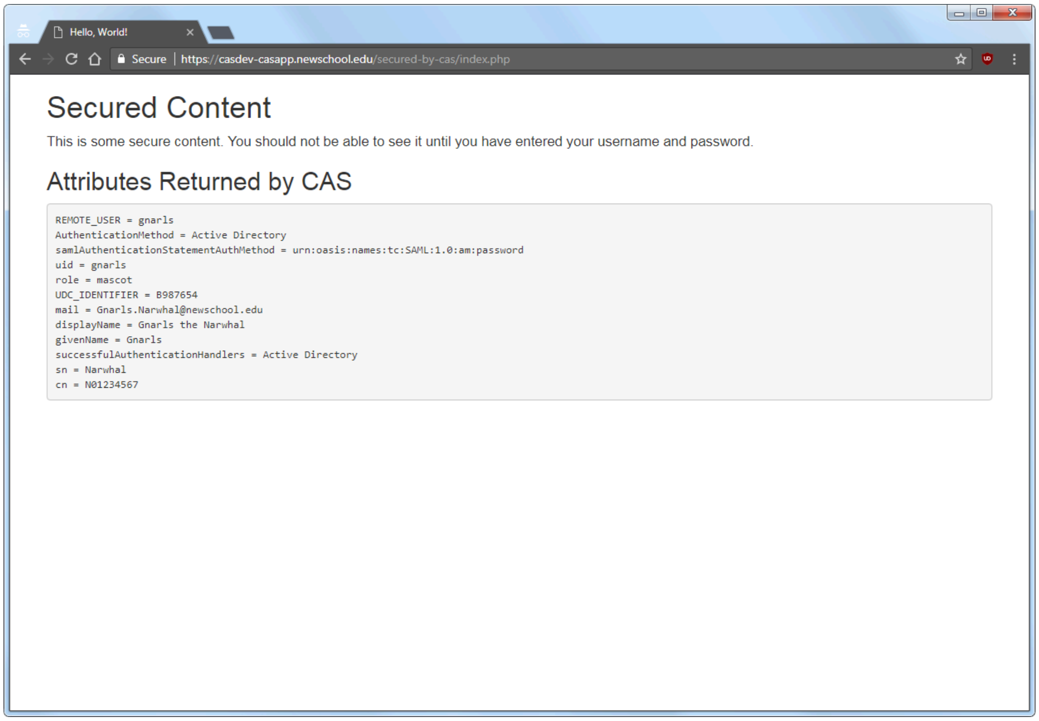


CAS development Apache mod_auth_cas server with username/password protection

For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!

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:


Hello, World!
Secure
https://casdev-casapp.newschool.edu/secured-by-cas/index.php

Secured Content

This is some secure content. You should not be able to see it until you have entered your username and password.

Attributes Returned by CAS

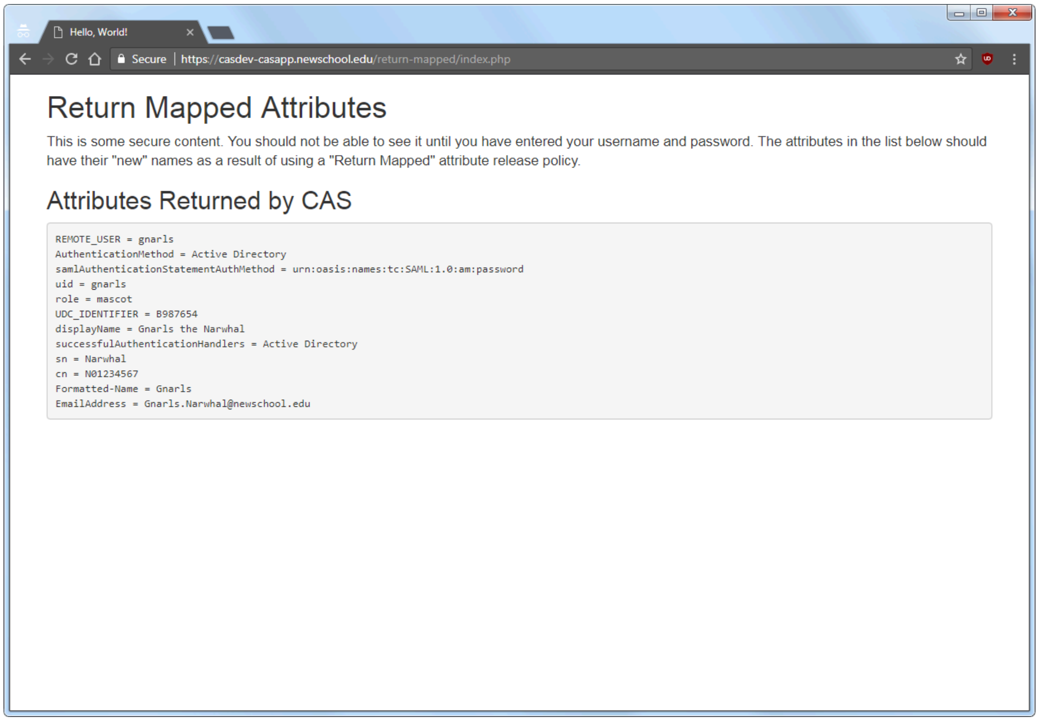
```
REMOTE_USER = gnarls
AuthenticationMethod = Active Directory
samlAuthenticationStatementAuthMethod = urn:oasis:names:tc:SAML:1.0:am:password
uid = gnarls
role = mascot
UDC_IDENTIFIER = B987654
mail = Gnarls.Narwhal@newschool.edu
displayName = Gnarls the Narwhal
givenName = Gnarls
successfulAuthenticationHandlers = Active Directory
sn = Narwhal
cn = N01234567
```

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 143) 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` :



Return Mapped Attributes

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.

Attributes Returned by CAS

```
REMOTE_USER = gnarls
AuthenticationMethod = Active Directory
samlAuthenticationStatementAuthMethod = urn:oasis:names:tc:SAML:1.0:am:password
uid = gnarls
role = mascot
UDC_IDENTIFIER = B987654
displayName = Gnarls the Narwhal
successfulAuthenticationHandlers = Active Directory
sn = Narwhal
cn = N01234567
Formatted-Name = Gnarls
EmailAddress = Gnarls.Narwhal@newschool.edu
```

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:** Multifactor 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 multifactor authentication (MFA) that supports multiple multifactor 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</artifactId>
        <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</artifactId>
        <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: Multifactor 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:      FeTtpcOFyDxvtrtOXqma74DztXf7INRDKENMhAOF
cas.authn.mfa.duo[0].duoApplicationKey: 3d787231f9b9e128a9b94647b6e968f1fe0deddc
```

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-scale=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.</big></p>
        <p><big>Click <a href="secured-by-cas/index.php">here</a> for some
          content secured by username and password.</big></p>
        <p><big>Click <a href="return-mapped/index.php">here</a> to see the
          results of the "Return Mapped" attribute release policy.</big></p>
        <p><big>Click <a href="secured-by-cas-duo/index.php">here</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: Multifactor 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.json 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 168).
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-cas-duo(\\z|/.*)",
  "name" : "Apache Secured By CAS and Duo",
  "id" : 1504200420,
  "description" : "CAS development Apache mod_auth_cas server with username/password and Duo MFA protection",
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePolicy"
  },
  "multifactorPolicy" : {
    "@class" : "org.apereo.cas.services.DefaultRegisteredServiceMultifactorPolicy",
    "multifactorAuthenticationProviders" : [ "java.util.LinkedHashSet", [ "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: Multifactor Authentication

Install and test the application

Before multifactor 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 98) (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 98):

```
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-install.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
```

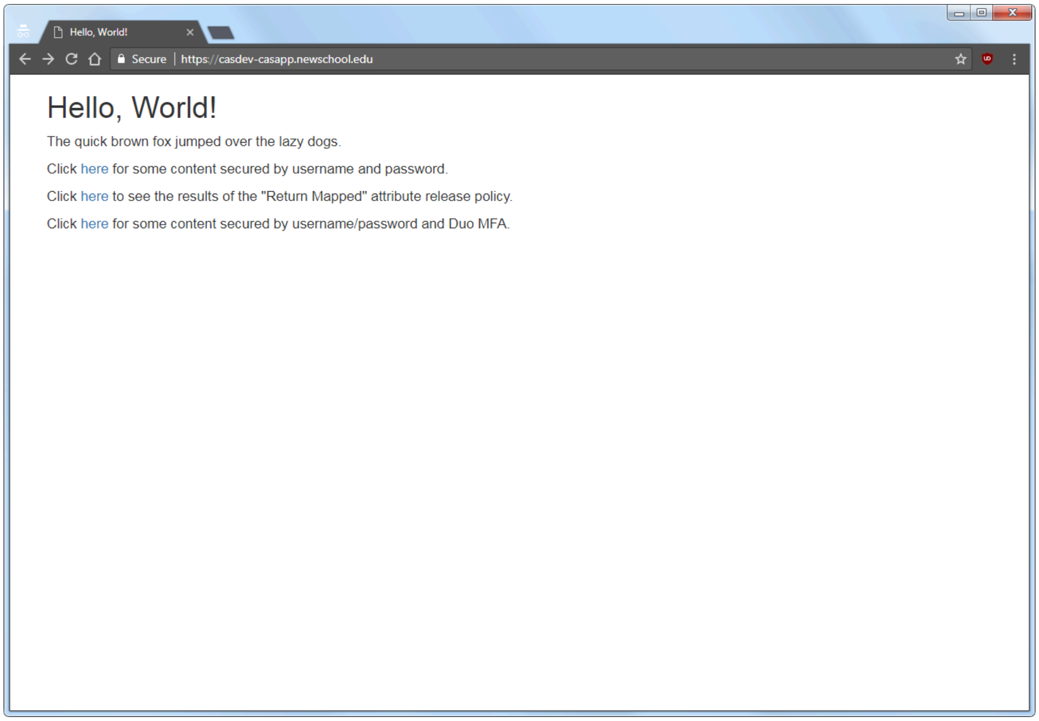
on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

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:



Hello, World!

The quick brown fox jumped over the lazy dogs.

Click here for some content secured by username and password.

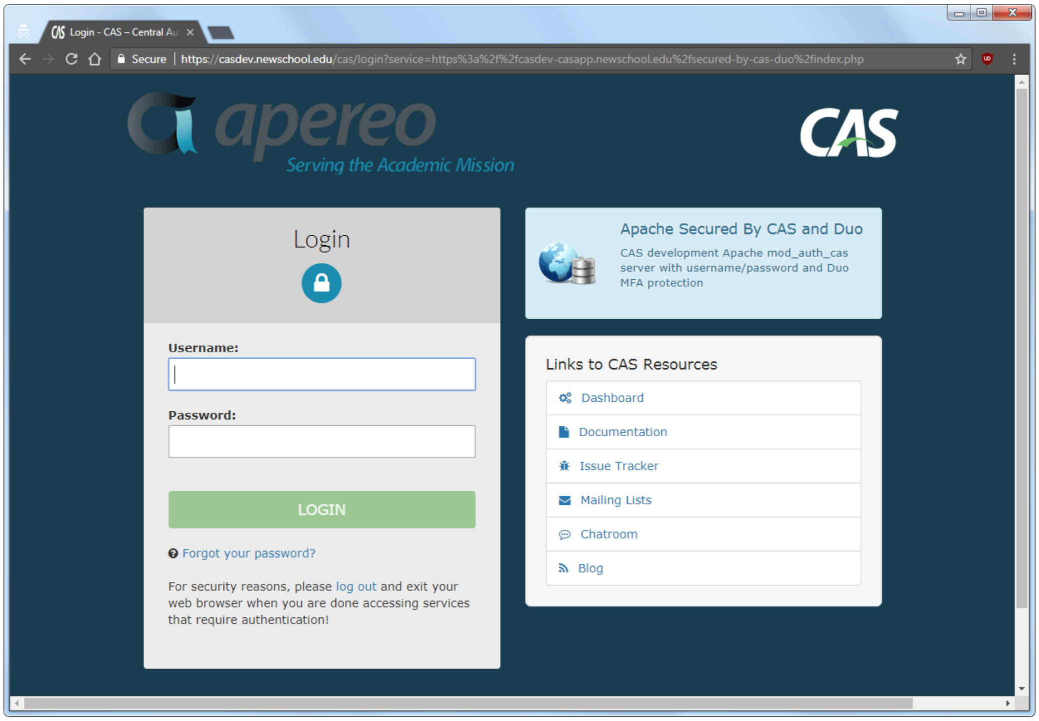
Click here to see the results of the "Return Mapped" attribute release policy.

Click here for some content secured by username/password and Duo MFA.

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:



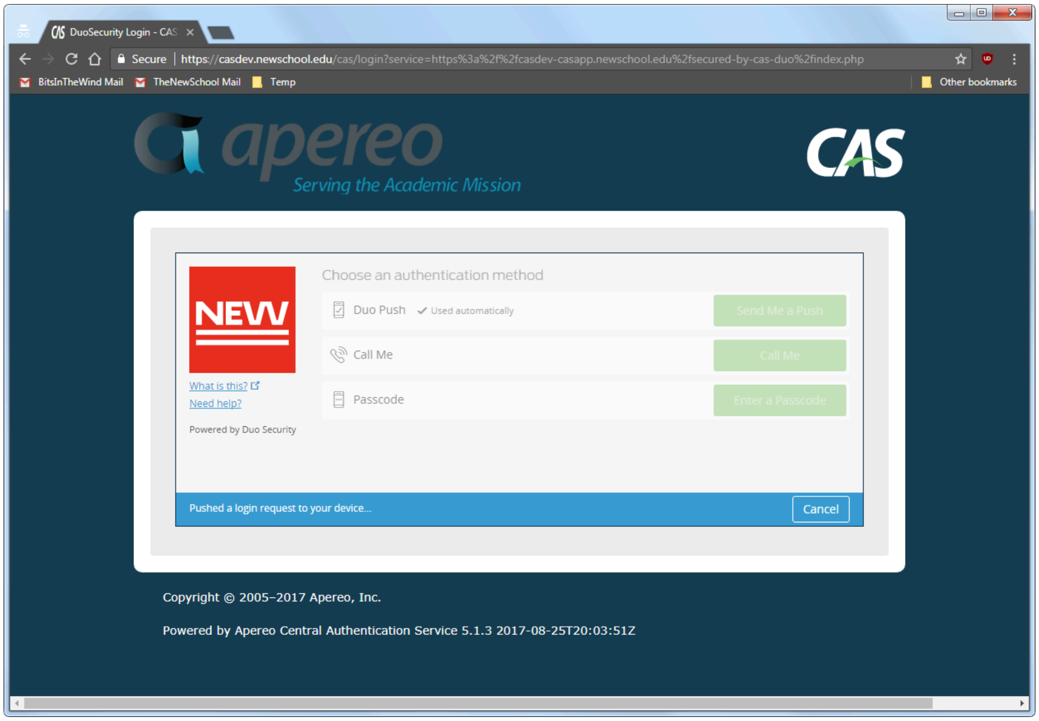
For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!

Apache Secured By CAS and Duo

CAS development Apache mod_auth_cas server with username/password and Duo MFA protection

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:

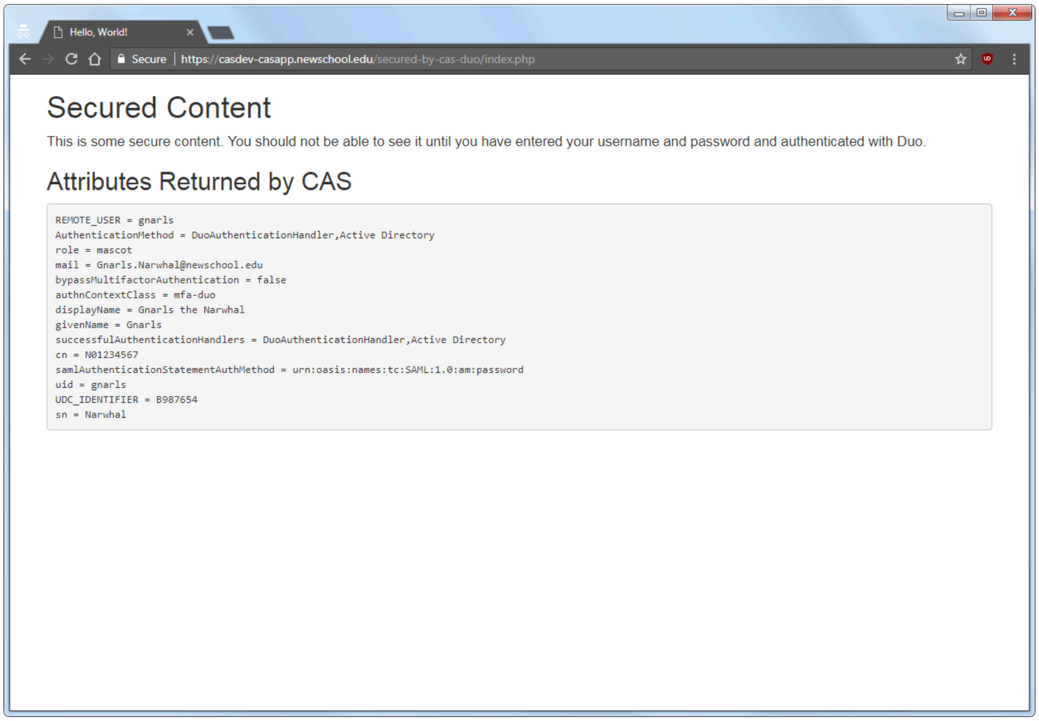


Copyright © 2005–2017 Apereo, Inc.

Powered by Apereo Central Authentication Service 5.1.3 2017-08-25T20:03:51Z

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:


Hello, World!
Secure
https://casdev-casapp.newschool.edu/secured-by-cas-duo/index.php

Secured Content

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.

Attributes Returned by CAS

```
REMOTE_USER = gnarls
AuthenticationMethod = DuoAuthenticationHandler,Active Directory
role = mascot
mail = Gnarls.Narwhal@newschool.edu
bypassMultifactorAuthentication = false
authnContextClass = mfa-duo
displayName = Gnarls the Narwhal
givenName = Gnarls
successfulAuthenticationHandlers = DuoAuthenticationHandler,Active Directory
cn = N01234567
samlAuthenticationStatementAuthMethod = urn:oasis:names:tc:SAML:1.0:am:password
uid = gnarls
UDC_IDENTIFIER = B987654
sn = Narwhal
```

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 multifactor 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 198) 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</artifactId>
        <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.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>
```

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 98) 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 services 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 services registry (page 106), 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-1503925297.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 initialization
   * 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 CAS server
   * doesn't have write access to the registry, then the save will fail and
   * the server will not start.
   *
   * To avoid that situation, and to make it clear that this endpoint is a
   * "desired" service, it is defined explicitly here.
   */
  "@class" :            "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" :         "https://casdev.newschool.edu/cas/idp/profile/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 98) (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/i
dp/metadata
<?xml version="1.0" encoding="UTF-8"?>
<EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata" xmln
s:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:shibmd="urn:mace:shib
boleth:metadata:1.0" xmlns:xml="http://www.w3.org/XML/1998/namespac
e" xmlns:mdui="urn:oasis:names:tc:SAML:metadata:ui" entityID="http
s://casdev.newschool.edu/idp">
    <IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:t
c:SAML:2.0:protocol urn:oasis:names:tc:SAML:1.1:protocol urn:mace:shi
bboleth:1.0">
        <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>
        <KeyDescriptor use="encryption">
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate>MIIDMjCCAhqgAwIBAgIVAMX1YMhKc
xKp/uraKDAmWA6u1AjnMA0GCSqGSIb3DQEB
CwUAMB8xHTAbBgNVBAMMFGNhc2Rldi5uZXdzY2hvb2wuZWR1MB4XDTE3MTAyMzE4
MzQxN1oXDTM3MTAyMzE4MzQxN1owHzEdMBsGA1UEAwwUY2FzZGV2Lm5ld3NjaG9v
bC5lZHUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCGo/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 198)), 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 98):

```
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-15090297
45.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-150902974
5.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 180), 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 https://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+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><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.

> ℹ **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.

> **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-scale=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.</big></p>
        <p><big>Click <a href="secured-by-saml/index.php">here</a> for 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-scale=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 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.

> **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:2.5.4.20"
    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="telephoneNumber"/>
```

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 143) 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="UDC_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 98) (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 98):

```
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
```

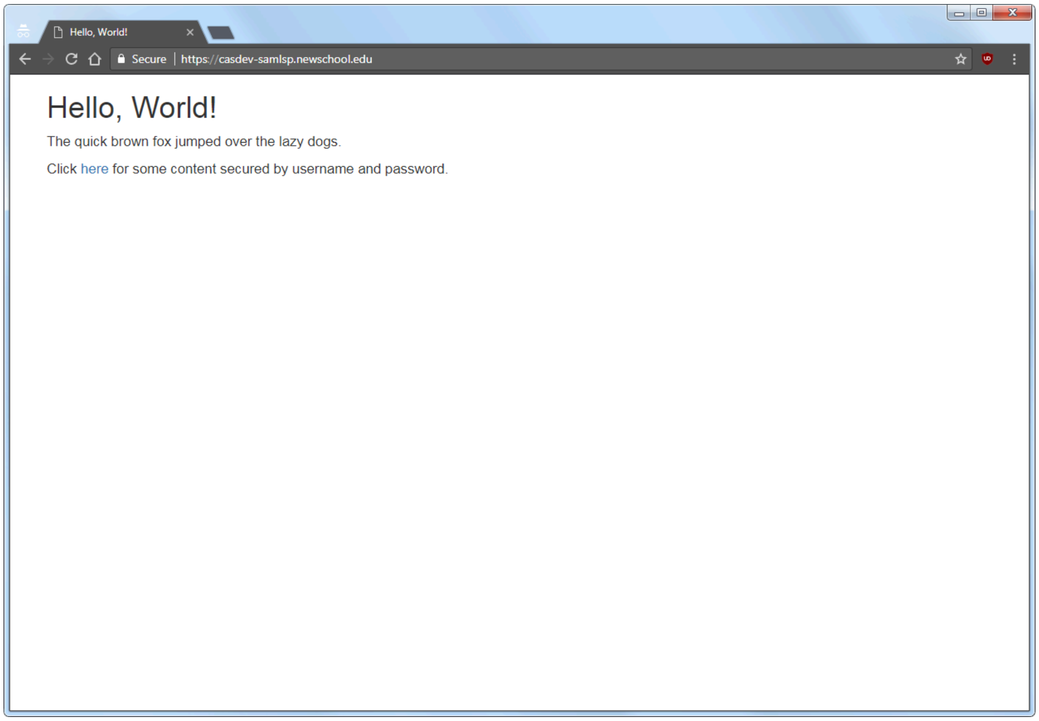
on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

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:



Hello, World!

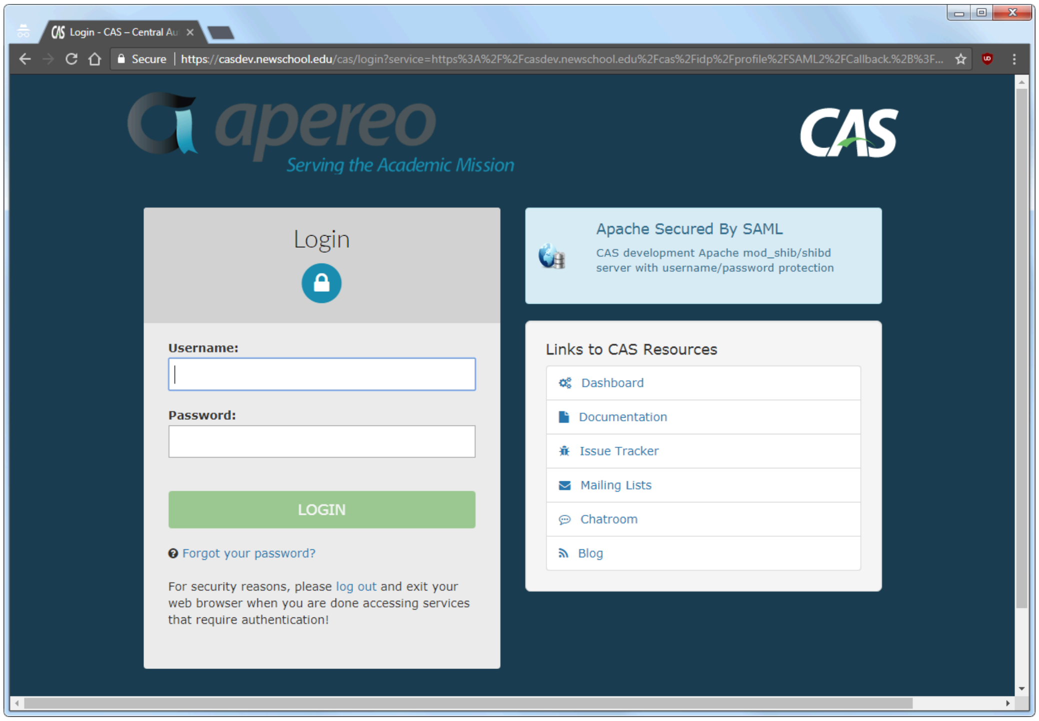
The quick brown fox jumped over the lazy dogs.

Click here for some content secured by username and password.

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:


Login - CAS – Central Au
Secure
https://casdev.newschool.edu/cas/login?service=https%3A%2F%2Fcasdev.newschool.edu%2Fcas%2Fidp%2Fprofile%2FSAML2%2FCallback.%2B%3F...
apereo
Serving the Academic Mission
CAS

Login

Username:

Password:

LOGIN

Forgot your password?

For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!

Apache Secured By SAML

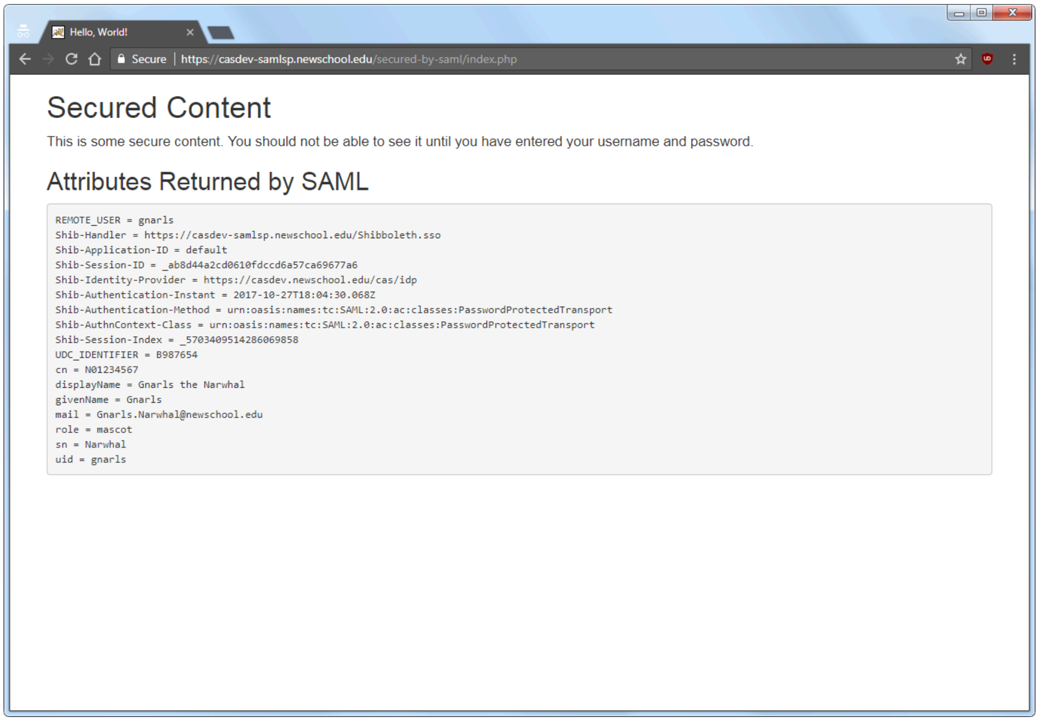
CAS development Apache mod_shib/shibd server with username/password protection

Links to CAS Resources

- Dashboard
- Documentation
- Issue Tracker
- Mailing Lists
- Chatroom
- Blog

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:


Hello, World!
Secure
https://casdev-samlsp.newschool.edu/secured-by-saml/index.php

Secured Content

This is some secure content. You should not be able to see it until you have entered your username and password.

Attributes Returned by SAML

```
REMOTE_USER = gnarls
Shib-Handler = https://casdev-samlsp.newschool.edu/Shibboleth.sso
Shib-Application-ID = default
Shib-Session-ID = _ab8d44a2cd0610fdccd6a57ca69677a6
Shib-Identity-Provider = https://casdev.newschool.edu/cas/idp
Shib-Authentication-Instant = 2017-10-27T18:04:30.068Z
Shib-Authentication-Method = urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
Shib-AuthnContext-Class = urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
Shib-Session-Index = _5703409514286069858
UDC_IDENTIFIER = B987654
cn = N01234567
displayName = Gnarls the Narwhal
givenName = Gnarls
mail = Gnarls.Narwhal@newschool.edu
role = mascot
sn = Narwhal
uid = gnarls
```

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 multifactor 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 multifactor authentication added:

```
{
  "@class" : "org.apereo.cas.support.saml.services.SamlRegisteredService",
  "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/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"
    }
  },
  "multifactorPolicy" : {
    "@class" : "org.apereo.cas.services.DefaultRegisteredServiceMultifactorPolicy",
    "multifactorAuthenticationProviders" : [ "java.util.LinkedHashSet", [ "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.DefaultRegisteredServiceMultifactorPolicy",
  "multifactorAuthenticationProviders" : [ "java.util.LinkedHashSet", [ "mfa-duo" ] ]
},
*/
```

References

- CAS 5: Duo Security Authentication
- CAS 5: Configuration Properties: Multifactor 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

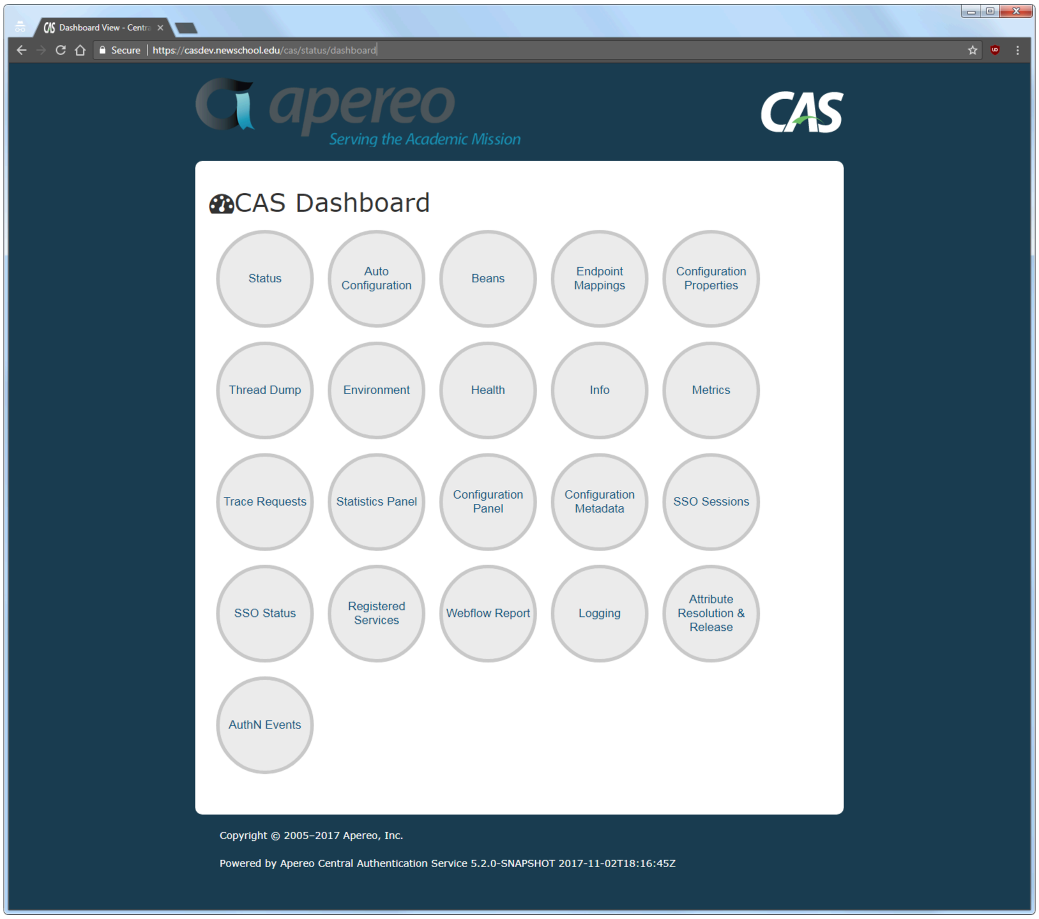
```
casdev-master# git add etc/cas/services/ApacheSecuredBySAML-1509030300.json
casdev-master# git commit -m "Set up SAML client"
[newschool-casdev 6ad660c] Set up SAML client
 1 file changed, 29 insertions(+)
 create mode 100644 etc/cas/services/ApacheSecuredBySAML-1509030300.json
casdev-master#
```

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

Enabling the dashboard (adminpages)

> **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.



Copyright © 2005–2017 Apereo, Inc.

Powered by Apereo Central Authentication Service 5.2.0-SNAPSHOT 2017-11-02T18:16:45Z

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 services management webapp (page 234) 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)$
```

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/dashboard
cas.adminPagesSecurity.users:           file:/etc/cas/config/admusers.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 /status/*
# endpoints ("adminpages").
#
# The syntax for each line is:
#
# username=password,grantedAuthority[,grantedAuthority][,enabled|disabled]
#
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 service management webapp (page 234), 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 `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/dashboard(\\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 98) (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 98):

```
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
```

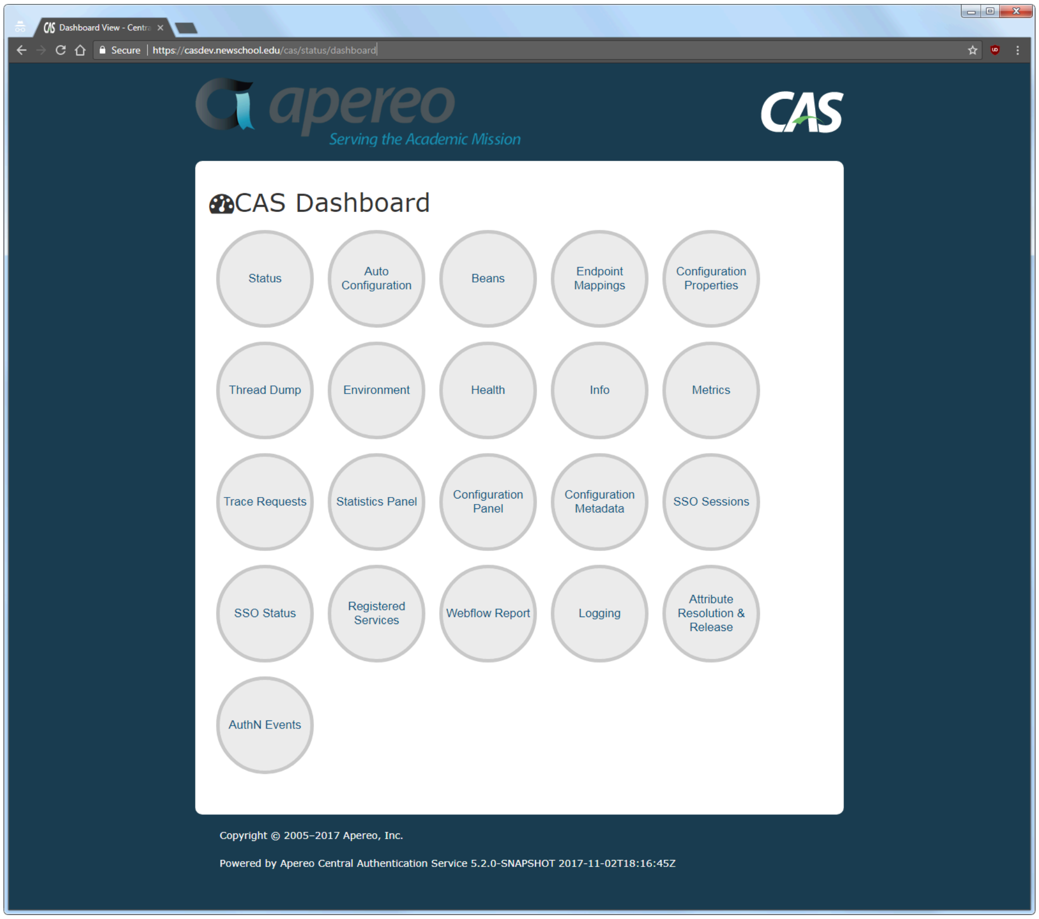
on all but one of the CAS servers (***casdev-srvXX***) to temporarily take those servers out of the pool.

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:



Copyright © 2005–2017 Apereo, Inc.

Powered by Apereo Central Authentication Service 5.2.0-SNAPSHOT 2017-11-02T18:16:45Z

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 65), 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 98), 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.

Building the services management webapp

> **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.

Customizing the CAS user interface

> **Summary:** The CAS server 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.

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.

High availability

> **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.

Building the configuration server

> **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 the service registry

> **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 the ticket registry

> **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.

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