SmarterApp Open Source Online Assessment System Deployment and Configuration Guide

Table of Contents

1 Revision History

Version	Comments	Author	Date
0.1 - **Draft**	Initial release - work in progress	Rami Levy Ajay Kumar	30 Oct 2014
0.2 - Draft	Updated Bootstrapping (Section 7). Updated PM config (6.1.3).	Rami Levy	06 Nov 2014
0.3 - Draft	Updated ProgMan parameter lists for ART, TSB, TIB, Progman, and Test Authoring to account for the newly protected MNA REST interface.	Rami Levy	20 Nov 2014
0.4 - Draft	Updated Portal configuration information (6.1.11). Updated Network firewall configuration (4.2.2).	Rami Levy	03 Dec 2014
0.5 - Draft	Added loadbalancer configuration information for PM, TA, ART, TSB, and MNA.	Rami Levy	10 Dec 2014
0.6 - Draft	Add Permissions spreadsheet link/ref	Rami Levy	16 Dec 2014
0.7 - Draft	Reformatting	Rami Levy	23 Dec 2014
1.0	Add sections on THSS, TIS, IRiS, and Item Scoring components. Updated Chapter 5 with Ansible-based environment configuration details.	Rami Levy	04 Feb 2015
1.1	Update section on Item Scoring. Clarify shared services deployment order (6.1.1). Additional cleanup in formatting and other minor corrections.	Rami Levy	15 Apr 2015
1.2	Update OpenAM Reference R13 to reflect v12 release. Updated Test Authoring config section. Added Dictionary, Student Report Processor, and CAT Simulator sections.	Rami Levy	10 Jul 2015
1.3	Minor clarification in 7.1.3 regarding protected roles. Added information regarding loading test packages into the TSB in 6.2.4.5.	Rami Levy	24 Jul 2015
1.4	Update 5.5.2 to include Spring security configuration. Update 7.1.2 to include	Rami Levy	5 Nov 2015

	information on default password and other user provisioning configurations.		
1.5	Add information on Uploading Accommodations and Designated Supports into ART (6.3.2.4)	Rami Levy	11 Nov 2015

2 References

Ref.	Reference	Author	Version
1	Assessment System Architecture and Technology Report http://www.smarterapp.org/architecture.html	Smarter Balanced	11 July 2014
2	Smarter Balanced Test Delivery Hosting Requirements http://www.smarterapp.org/specifications.html	Smarter Balanced	V2 (01 May 2014)
3	Smarter Balanced Test Delivery Hosting Cost Calculator http://www.smarterapp.org/specifications.html	Smarter Balanced	V2 (01 May 2014)
4	Smarter Balanced Component Dependencies Matrix (*Administrative* repository in Bitbucket: guide/Component Runtime Dependencies.pdf)	Smarter Balanced	Latest
5	SBAC SSO Component Design Document http://www.smarterapp.org/documents/SmarterBalanced_ArchitectureReport_07112014.pdf	Identity Fusion, Inc.	1.1 (11 Feb 2014)
6	Firewall Network Matrix reference (*Administrative* repository in Bitbucket: guide/firewall-network-matrix.pdf)	Smarter Balanced	Latest
7	Permissions, Roles, and Component mappings (Permissions repository in Bitbucket: Documents/permissions-roles-components.xlsx)	Smarter Balanced	Latest

Ref.	Repository Reference	URL
R1	Administrative	https://bitbucket.org/sbacoss/administrative_release
R2	Program Management (PM)	https://bitbucket.org/sbacoss/programmanagement_release
R3	Permissions	https://bitbucket.org/sbacoss/permissions_release
R4	Monitoring and Alerting (MNA)	https://bitbucket.org/sbacoss/monitoringandalerting_release
R5	Core Standards (CS)	https://bitbucket.org/sbacoss/corestandards_release

R6	Test Authoring	https://bitbucket.org/sbacoss/testauthoring_release
R7	Test Spec Bank (TSB)	https://bitbucket.org/sbacoss/testspecbank_release
R8	Test item Bank (TIB)	https://bitbucket.org/sbacoss/testitembank_release
R9	Administration and Registration Tools (ART)	https://bitbucket.org/sbacoss/adminandregtools_release
R10	Test Delivery (proctor)	https://bitbucket.org/sbacoss/tds_release
R11	Student	https://bitbucket.org/sbacoss/student_release
R12	Portal	https://bitbucket.org/sbacoss/portal_release
R13	OpenAM	https://bitbucket.org/sbacoss/openam12_release
R14	OpenDJ	https://bitbucket.org/sbacoss/opendj_release
R15	Shared Security	https://bitbucket.org/sbacoss/sharedsecurity_release
R16	Test Integration System (TIS)	https://bitbucket.org/sbacoss/testintegrationsystem_release
R17	Teacher Hand Scoring System (THSS)	https://bitbucket.org/sbacoss/teacherhandscoresys_release
R18	Item Scoring Engine	https://bitbucket.org/sbacoss/itemscoring_release
R19	Dictionary	https://bitbucket.org/sbacoss/dictionary_release
R20	CAT Simulator	https://bitbucket.org/sbacoss/catsimulator_release
R21	Student Report Processor	https://bitbucket.org/sbacoss/studentreportprocessor_release

3 Introduction

This guide provides instructions on how to deploy the Smarter Balanced Open Source Online Assessment System. Where appropriate, instructions are made to be hosting-environment independent.

3.1 Target Audience

This guide is primarily intended for system administrators.

3.2 How to Use This Guide

If you are new to the Smarter Balanced system, begin with the Architectural Overview below. If you know which components you intend to deploy, begin at *Section 4 (Initial Considerations)* of this document and follow each step there. Continue to *Section 5 (Server Creation)*. For most component deployments, Shared Service components will be required. Use the chart in *Section 6 (Component-Specific Deployments)* to determine which components are needed and follow the instructions in those sections, which will take you through post-deployment scenarios. Each server you create follows the steps outlined in Section 5, and is then customized by the steps in Section 6 (component-specific deployment).

3.3 Architectural Overview

The system's overall architecture is depicted below. Additional details are available at [1]. This Guide will reference the groupings depicted in this Figure that relate to Smarter Balanced Contract 11. This ***excludes*** the following components:

- Digital Library
- Item Authoring and Item Pool (IAIP), also known as Item Bank
- Reporting
- Data Warehouse

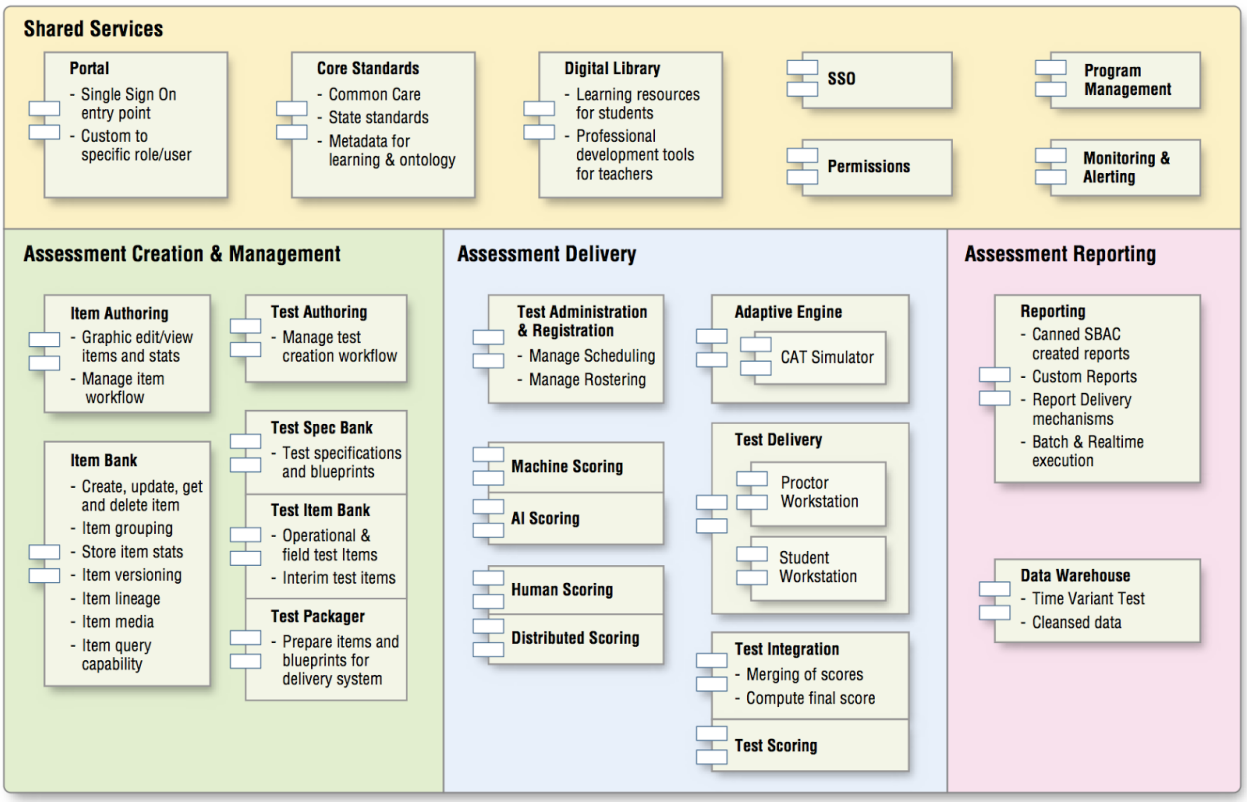


Figure 1: Smarter Balanced Architecture

4 Initial Considerations

4.1 Infrastructure Selection

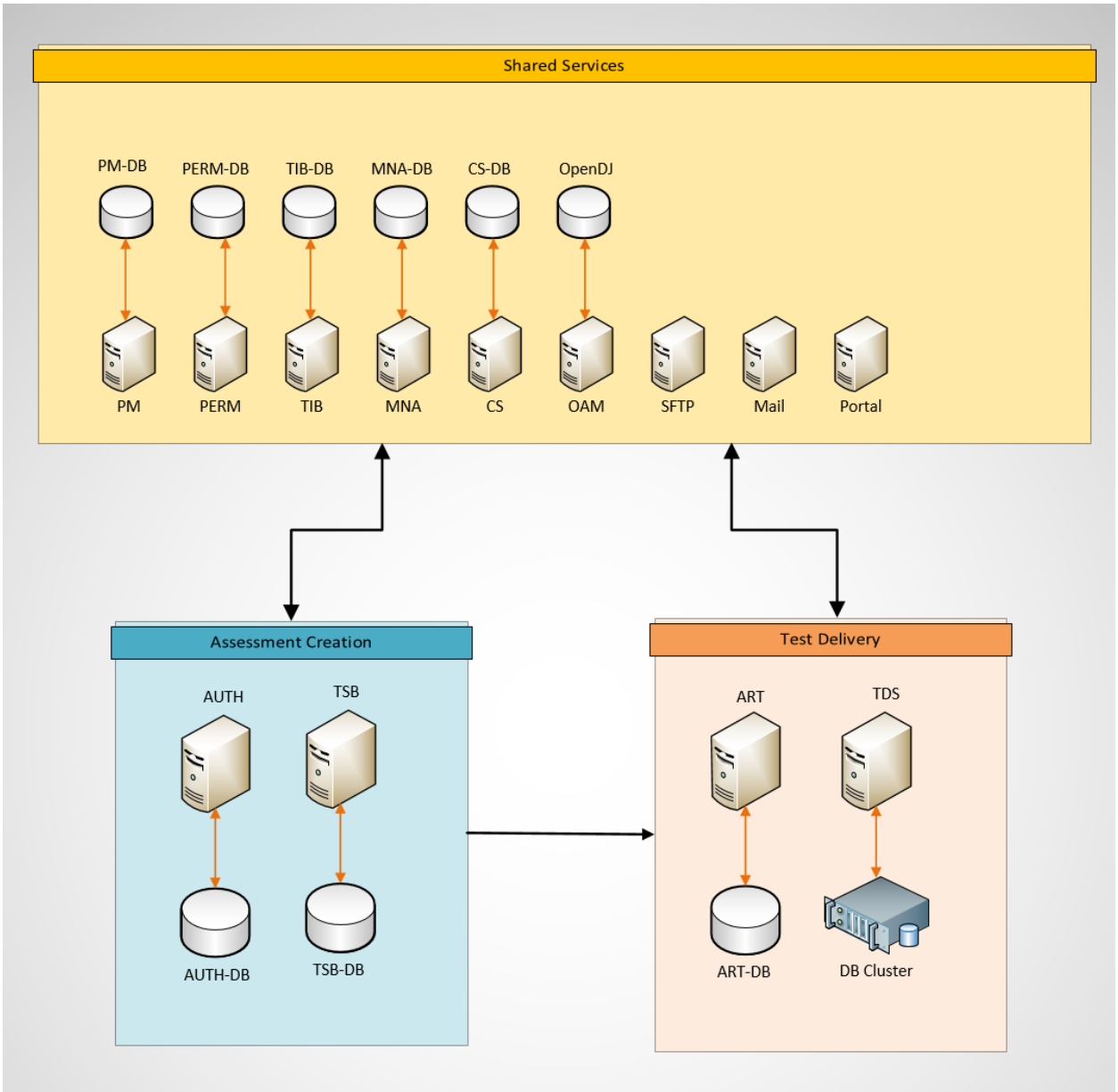
When selecting a target deployment infrastructure, there are a number of considerations to make, including performance requirements, cost, support, availability, and so forth. Examples of cloud-based infrastructure options are Amazon Web Services (AWS) and Rackspace. Alternatively, it would be possible to deploy to an in-house data center with physical rather than virtual hardware. Hybrid solutions are also possible; however for performance reasons it is strongly advised to co-locate all components within the same network wherever possible. Please refer to the *Deployment Assumptions* and *Deployment Configurations* sections in [2] for more information. Cost considerations for an AWS-based deployment are described in [3].

4.2 Server Instances and Network Topology Selection

Once the infrastructure is selected, the machine instance types, database configurations, and deployment node structures (e.g. load balancing) must be defined - again based on the deployer's specific requirements. Please refer to the *Test Delivery Unit* section in [2] for more information.

4.2.1 Example Topology

In one example implementation shown below, each service is deployed onto its own server, and connected with a dedicated database server. The TDS web server is attached to a DB cluster service (e.g. RDS). The TDS server could also be placed behind a load balancer, as can any other service.



4.2.2 Network Security / Access

For security purposes, it is recommended that a single machine be used as the entry point (via SSH) for administrators to manage all machines. The security groups on the remaining machines should be configured such that external access via ssh is blocked. Specifically, the configuration provided in [6] is recommended.

4.3 Domain Name Selection

It is recommended that all deployments in a single environment use the same domain name, to reduce complexity. The selection of a domain name (e.g. my.state.edu) also allows for the

selection of a proper SSL wildcard certificate (see next section). If deployed in AWS, then you need to use the AWS/Route 53 dashboard to register the DNS Domain with AWS.

4.4 CA-Signed SSL Certificate

A Certificate Authority issues signed digital certificates which certify the ownership of a public key by the named subject of the certificate. Once a domain name is selected, it is strongly advised to purchase a *wildcard certificate* that will encompass your domain name.

5 Server Creation / Configuration / Provisioning Guide

Server creation is specific to the environment in which the components are being deployed. The instructions here will use Amazon (AWS) as an example; other environments should generally follow the same steps.

5.1 Assumptions

This guide assumes the administrator has full access rights to the target deployment environment.

5.2 Checklist

At this point, the following choices should have been made:

1. Deployment infrastructure (e.g. AWS);
2. Network topology (e.g. load balancers / HA servers);
3. Wildcard SSL certificates for the selected domain name;
4. Instance types for each component being deployed (e.g. AWS EC2 m3.large instance type for Test Authoring component) including databases;
5. Required (dependent) components for all components being deployed (see [4]).
6. Register domain name with AWS's Route53.

For each of the components being deployed, follow the appropriate guidelines in Section 6 (Component-Specific Deployments).

5.3 Linux Environment Configuration

5.3.1 Prerequisites

The first server that goes online and will be used for the other server creations must be called ssh-<env>.<domain>. Its security group is likewise called ssh-<env>.<domain> with SSH opened. This server must be updated correctly in AWS's Route 53 and AWS security groups. Basically, the AWS security groups will reference this server to configure access. You must be able to ssh as ubuntu to the server being created. Once this server is created, execute the following commands as root:

```
apt-get -y install mercurial
apt-get -y install acl
```

Create $HOME/.hgrc as

```
[ui]
username = <First> <Last> <Email>
```

Sample .hgrc file:

```
[ui]
username = Test Sbac <TestSbac@bitbucket.org>
```

Create AWS keypair:

```
EC2 Dashboard -> Network and Security -> Key Pairs -> Create
```

Save the key as a file and name for later use and updates. The key pem file should be saved under `/root/.ssh/<keyname>.pem`. File permissions should be 600.

5.3.2 Download the Source

It is important to follow this procedure to obtain the source; without the proper ACLs, the scripts will fail:

1. `hg clone https://<YOUR ID>@bitbucket.org/sbacoss/administrative_release`
2. `cd administrative_release`
3. `setfacl --restore=.acl`

If and when you need to check in any changes, you'll need to regenerate the .acl file first:

1. go to the `environment` directory
2. `getfacl -R . > .acl`
3. Perform your normal procedure for checking in your changes to your repository.

5.3.3 Define Environment's Servers

The file `machines_and_types.xls` contains a list of servers, their role in the environment, what size machine they are, what the software is that they run. This file needs to be **edited** to reflect your environment and then **saved** as a comma separated value (CSV) text file named `machine_configs.txt`.

5.3.4 Define Passwords and Credentials

Create a file on your local machine at the path `/etc/ansible` called `external_vars.yml`. It contains a set of key/value pairs that must be filled in to properly set certain credentials in your configurations (an example file is included). For best results, complete this file first before proceeding to create the environment. Ansible will automatically replace these vars when it executes. Sample file YML with required variables:

```
---
mysql_password: TEST001
tomcat_password: TEST002
DOMAIN_NAME: test.org
sasl_password: testSALSpassowrd0101
admin_email: tester@test.org
ssmtp_server_and_port: mail.test.org:10025
ssmtp_domainname: test.org
```

We were not able to use this methodology to replace all files. There are specific files defined in the ansible directory. These files must be replaced with site-specific values prior to running the create_instance.sh or any ansible command. The file listing is in the ansible section.

5.3.5 Define Firewall Configuration

In order to secure access to the proper ports and services on each machine and limit machine exposure to the bare minimum, a matrix is used to document this. The file `Firewall_dependencies.xls` contains the firewall settings of all the machines. Once this

matrix is edited to reflect your environment, save it as a comma separated value (CSV) text file called `firewalls.txt`.

5.3.6 Ansible and AWS Prerequisites

Ansible is an open-source software platform for configuring and managing computers. It combines multi-node software deployment, ad hoc task execution, and configuration management. It manages nodes over SSH using ssh key files. Ansible is meant to be run at anytime and not cause any changes to servers that are already configured per the rules. Information about ansible can be found here: http://docs.ansible.com/intro.html

Ansible has a concept of "best practices" (http://docs.ansible.com/playbooks_best_practices.html). This includes the directory structure, file naming, separating the different environments, etc. We have adhered to these practices.

Our ansible configuration requires the AWS Ubuntu Linux client to be installed. The following commands will take care of the base software install. Mercurial is needed to access the sbac repository.

```
apt-get -y install software-properties-common
apt-add-repository ppa:ansible/ansible
apt-get update
apt-get -y install ansible
apt-get -y install awscli
apt-get -y install python-dev
apt-get -y install python-setuptools
apt-get -y install python-boto
easy_install pip
pip install awsebcli
```

The default ansible user for ssh is root, for AWS/Ubuntu it is ubuntu. Edit `/etc/ansible/ansible.cfg`

- Uncomment and change `remote_user=root` to `remote_user=ubuntu`.
- Uncomment `host_key_checking = False`. Ansible/SSH installation will fail if this is not in sync.

You can run the `eb --help` command to verify that the AWS client has installed.
You will need to configure the AWS client prior to running ansible. Basically, AWS/EBS controls the inventory/hosts part of ansible.

AWS uses a set of environmental variables stored in `/usr/local/etc/ec2.env`. This file should have root.root/600 permissions. Here is a sample:

```
export AWS_ACCESS_KEY=<YOUR KEY>
export AWS_ACCESS_KEY_ID=<YOUR KEY>
export AWS_SECRET_KEY=<YOUR KEY>
export AWS_SECRET_ACCESS_KEY=<YOUR KEY>
```

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export AWS_DEFAULT_REGION=us-east-1
export AWS_DEFAULT_OUTPUT=text
```

The AWS dynamic inventory script called `ec2.py` is used to generate information about the currently running servers. This allows the use of the tags populated in EC2 to apply the software for certain roles. For example, to update only the servers with the Tag of *Type* set to tomcat, the command would be:

```
ansible-playbook -s -i ec2.py site.yml --limit tag_Type_tomcat
```

The following roles are currently defined:
`common, Core_Standards, mail, mongo, mysql, sftp, ssh, ssmtp, and tomcat`

`common` is applied to all machines in the environment. The other roles are applied based on the server files rules.

The main playbook, `site.yml`, should contain all of the files that have the server role rules and looks like:

```
---
- include: tomcat_servers.yml
- include: mongo_servers.yml
- include: mysql_servers.yml
- include: Core_Standards_servers.yml
- include: mail_servers.yml
- include: ssh_servers.yml
- include: sftp_servers.yml
- include: oam_servers.yml
- include: odj_servers.yml
  sudo: yes
```

Each of those files contains the individual rules for that server. Looking at the `tomcat.yml` file:

```
---
- hosts: tag_Type_tomcat
  roles:
    - common
    - ssmtp
    - tomcat
```

It says that all hosts that have the tag Type with value of tomcat will match and have those roles. Each of those roles will be in the ansible/roles directory structure:

```
drwxr-xr-x 5 root root 4096 Oct 20 11:16 ansible/roles/common
drwxr-xr-x 3 root root 4096 Dec 22 15:40 ansible/roles/ssmtp
drwxr-xr-x 6 root root 4096 Nov 18 13:50 ansible/roles/tomcat
```

and each of these directories will have some (or all of these standard directories):

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
```

when running ansible-playbook the output is color coded. Green means no changes, yellow means a rule was applied, and red is error. Sample output:

```
ansible-playbook -s -i ec2.py site.yml --limit
mail-demo.opentestsystem.org
```

In order for ansible to be portable to other environments, passwords and private information is kept in the `/etc/ansible/external_vars.yml` file. Currently the contents are:

```
---
mysql_password: XXXXX
tomcat_password: XXXXX
DOMAIN_NAME: XXXXX
sasl_password: XXXXX
admin_email: XXXXX
ssmtp_server_and_port: XXXXX
ssmtp_domainname: XXXXX
```

Just replace XXXXX with the values appropriate for your environment.

Additionally, the following site specific files must be updated prior to the environment creation.

```
./environment/ansible/roles/tomcat/files/.ssh/tomcat.ENVNAME.keystore
./environment/ansible/roles/tomcat/templates/server.xml.j2
./certs/tomcat.ENVNAME.keystore
./certs/ci/tomcat.ENVNAME.keystore
```

`./certs/.truststore.aws`

These files contain the Tomcat SSL keystores.

5.3.7 Environment Creation

For each of the server roles in machine_configs.txt a server needs to be created. These can be combined, split up, or multiple servers created and put behind a load balancer. Once the servers are up and running with Ubuntu, then ansible can be run to load all of the appropriate software for that server's role.

A series of scripts and an ansible directory structure have been created that can automatically generate a fully functional environment. The flow of the environment creation is as follows:

1. Our AWS networking configuration uses EC2, not VPC. We use dynamic DNS, aka AWS's Route53, to reconcile the issue of reboots and access. Edit the values for Domain and ZoneID in ./environment/create_instance_update_route53_cname.sh . Copy the edited file to /usr/local/bin. You will also need to copy ./environment/update_route53_cname.conf to /etc/init
2. Create the first server as described in the "Prerequisites" section in this chapter.
3. Change directory to "`environment`" where the `create_env.sh` exists. This is a template script; you will need to change env, domain, key, etc. to reflect your specific configuration. This script contains the individual lines necessary to create the servers. It calls a script called ./create_instance.sh for each server. This is a sample file which needs to be edited to match your ansible configuration. It assumes that the ansible directory is located in the subdirectory relative to ./create_instance.sh .
4. create_instance.sh requires 3 arguments to be passed in:
    - **-n** - the fully qualified domain name (FQDN) of the server
    - **-e** - what environment the server will run in (this can be any word. It will be used as a tag in AWS EC2 instances). This is also configured in ansible (see below).
    - **-r** - what role the machine has in the environment. Based on the role the server has, it can figure out what type of server it is (tomcat, mysql, mongo, etc.)
    - **-k** - keypair name, generated in AWS, test file version stored in /root/.ssh/id_rsa
    - **-d** - DNS domainname

    An example of a call:

    ```
    create_instance.sh -n sftp-demo.opentestsystem.org -e demo -r sftp -k
    demotest -d opentestsystem.org
    ```

    The logical flow of `create_instance.sh`:

    1. Read and parse `machine_configs.txt` looking for hardware configuration for the role being selected.
    2. Read and parse `firewalls.txt` to find out what machines are allowed to talk to it and on what ports they want to talk. If a machine needs to open up several ports - those values should be separated by '/'. For example if a server needs to open up SSH and HTTP for a server - the field for that should look like

SSH/HTTP. There is one AWS security group configured per server, for maximum flexibility.

3. For each of the servers that need access - the script make sure a security group exists for that server and then adds that server to the ingress list.
4. The server is then created from a default Ubuntu image.
5. Once the server is fully up the EC2 tags are populated. The tags include the role (*Role*), the environment (*Env*), and type (*Type*). These fields are used extensively in ansible to target what software should be on them. The Env file is stored in group_vars/tag_Env_<YOUR ENV>. This file will be generated with default values if it does not already exist. A sample file would look like this:

```
---
ssmtp_server: mail.<YOUR ENV>.<YOUR DOMAIN>
ssmtp_server_and_port: mail-<YOUR ENV>.<YOUR  DOMAIN>:10025
ssmtp_domainname: <YOUR DOMAIN>
```

6. The DNS information is populated in Route53. This allows machines to be addressed with their FQDN.
7. The machine is then configured with ansible.

5.4 Component Software Provisioning and Deployment

Provisioning a server essentially comprises these steps:

1. Deploying the component software package itself (e.g., .war file(s)).
2. Configuring the system and specific installed components. For example, connecting the component to any number of shared services; setting server startup parameters, and setting component initialization parameters via Program Management.
3. Creating, installing, and populating databases for components where applicable.

The last two items will be covered in the *Component-Specific Deployments* section of this document.

5.5 Tomcat Webserver SSL Certificate Installation

> ***NOTE:*** *SSL certificates may either be installed on either a load balancer (if used), or on the individual web servers. The information below describes how to do this on individual servers.*

The procedure for SSL certificate installation is common for all web server application components. The following steps are used to create and install SSL certs once they have been issued by a CA.

5.5.1 Certificate Installation

Add the originally created private key to a JKS keystore, which will be used for all servers. Place this keystore in a secure directory (usually `$CATALINA_BASE`/.ssh) with the appropriate restrictions for private keys (e.g. chmod 600, ownership by tomcat only). For testing purposes, a self-signed certificate may be added, and replaced later by the CA-signed cert. Run the script

`create_tomcat_certs.sh --help` (found at [R1]) for more information. Essentially this is what needs to happen:

1. Create a keystore using the (self-signed or CA-signed) private key
2. Import the trusted cert from the keystore into a trust store
3. Save both of these to the secure directory

5.5.2 Spring Security Certificate

Since all Tomcat applications in this system use Spring security, the samlKeystore.jks file should be installed at `$CATALINA_BASE/resources/security/`. This keystore file should include the SSL certificate you have obtained for your system as well.

5.5.3 Configuring the SSL Connector

> ***NOTE**: This step is automatically performed by the transform_server.sh script discussed above. You should only need to modify the values shown in red below to match your specific configuration.*

Tomcat's global connector options are configured in Tomcat's configuration file, located at `$CATALINA_BASE/conf/server.xml`. Modify it by adding a connector element to support for SSL or https connection. Add a new connector port 8443 section by copying the existing commented `Connector port="8443"` or uncomment and use the same section as a new connector port. Finally, add the keystore file, key alias server name and replace key store password with yours. Save your changes and restart the server to make changes effective. Your connector port should look similar to the section below:

```
<Connector
        protocol="org.apache.coyote.http11.Http11NioProtocol"
        port="8443" maxThreads="200" maxHttpHeaderSize="16384"
        scheme="https" secure="true" SSLEnabled="true"
        keystoreFile="/path/to/keystore/file" keystorePass="your_keystore_password"
        keyAlias="your_server_key_name" clientAuth="false" sslProtocol="TLS"/>
```

5.5.4 Forcing HTTPS-only Connections in Tomcat

In order to prevent non-secure HTTP access, the following steps should be performed.

- In $CATALINA_HOME/webapps/application/WEB-INF/web.xml, add this after </servlet-mapping>, add the following code. **NOTE** that this one step must be done after every tomcat **web-application** deployment.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Everything in the webapp</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

```
    </user-data-constraint>
  </security-constraint>
```

- Enable the load balancer to recognize server side HTTP / HTTPS request or redirects.
- In server.xml add this at the end, before </Host> :

```
<Valve className="org.apache.catalina.valves.RemoteIpValve"
        remoteIPHeader="X-Forwarded-For"
        remoteIPProxiesHeader="X-Forwarded-By"
        protocolHeader="X-Forwarded-Proto" />
```

- NOTE: Make sure the tomcat connector is redirecting 8080 to 8443. Example:

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" maxHttpHeaderSize="16384" />
```

5.6 Windows Server Configuration

Unlike the other open source applications, Teacher Hand scoring System (THSS) and Test Integration System (TIS) are developed in C-Sharp and require Windows-based machines.

5.6.1 Server Configuration

To create a server, follow the following steps:

1. Select an AMI from the publicly available AMIs. Select either Microsoft Windows Server 2008 R2 Base or Microsoft Windows Server 2008 R2 with SQL Server Standard.
2. After selecting an appropriate instance type, select an availability zone (best is to keep the servers in the same AZ). It's recommended to enable termination protection as well.
3. Then select an appropriately-sized root partition, and add additional storage if needed.
4. Create a new (or select an existing) security group for this server.
5. You will create a new (or select existing) PEM Key for access. Once the server is created and launched, record the Administrator Password/key for RDP access.
6. RDP into console as Administrator and configure Virus software (ie. Microsoft Essentials), Windows updates (Automatically), firewall ports (22, 80, 443, 1433, 3389). Make sure that these ports map to the same ports in the AWS security group.
7. Add local administrator accounts to both Administrator group and RDP group.

5.6.1.1 IIS Configuration

Configure the IIS webserver to access the application.

5.6.1.2 SSL Certificate Installation

Follow the below steps to configure the SSL certificates:

1. Click **Start**, mouse-over **Administrative Tools**, and then click **Internet Services Manager**.
2. In the **Internet Information Services (IIS) Manager** window, select your server.
3. Scroll to the bottom, and then double-click **Server Certificates**.
4. From the **Actions** panel on the right, click **Complete Certificate Request** and locate your certificate file

5. In the **Open** window, select ***.*** as your file name extension, select your certificate (it might be saved as a .txt, .cer, or .crt), and then click **Open**.
6. In the **Complete Certificate Request** window, enter a **Friendly name** for the certificate file, and then click **OK**.
7. In the **Internet Information Services (IIS) Manager** window, select the name of the server where you installed the certificate.
8. Click + beside **Sites**, select the site to secure with the SSL certificate.
9. In the **Actions** panel on the right, click **Bindings.**
10. **Click Add and In the Add Site Binding window: follow this**
    a. For Type, select https.
    b. For IP address, select All Unassigned, or the IP address of the site.
    c. For Port, type 443
    d. For SSL Certificate, select the SSL certificate you just installed, and then click OK
11. Close the Site Bindings window.
12. Close the Internet Information Services (IIS) Manager window. Your SSL certificate installation is complete.

Tag the instance by setting the Name key to the exact DNS name required for the instance you're creating, making sure it follows the wildcard pattern for your certificate.

5.6.1.3 DNS Configuration

There is no route53 update script available for Windows. Basically, every reboot would require security group and route53 updates for new IPs. We recommend installing into a VPC with static IPs. This will allow for security group access from the webservers to database server.

5.7 Database Installation notes

Applications will require a database backend. While the application will create the basic table structure, it will not create the database itself. The following are examples of creating a database for mysql and mongodb.

5.7.1 MySQL Database Creation

```
$ mysql --user=root -p <YOUR PASSWORD>
create database permissions
ns_db;
create user '<YOUR USERNAME>'@'%' identified by '<YOUR PASSWORD>';
grant all on permissions_db.* to '<YOUR USERNAME>'@'%';
flush privileges;
```

5.7.2 MongoDB Database Creation

```
$ mongo -u <YOUR USERNAME> -p <YOUR PASSWORD> --authenticationDatabase admin
use admin;
db.addUser( {
       user: "<YOUR USERNAME>",
       pwd: "<YOUR PASSWORD>",
```

```
      roles: [ "dbAdminAnyDatabase", "userAdminAnyDatabase", "clusterAdmin",
"readWrite"  ]
      } );

use <APPLICATION DATABASE>;
db.addUser(
  {
      user: "<YOUR DB USERNAME>",
      pwd: "<YOUR PASSWORD>",
      roles: [ "readWrite" ]
  } );
```

6 Component-Specific Deployments

Components/servers should be created/deployed in the order listed in this chapter. For example, if deploying Monitoring and Alerting, its dependencies (shown in Table 1) should be created and deployed as follows: SSO, PM (read the MNA column from top down).

6.1 Shared Services Deployment

These components support the other components and create a cohesive system for the end users. Most components require the installation of the shared services components. Refer to [4] and the shared services chart below for details.

Deployed Component → / Dependent component ↓	Single Sign On (SSO)	Program Management (PM)	Permissions	Monitoring & Alerting (MNA)	Core Standards (CS)	Portal
Single Sign On (SSO)	n/a	Y	Y	Y	Y	Y
Program Management (PM)		n/a	Y	Y	Y	Y
Permissions			n/a	Y		Y
Monitoring & Alerting (MNA)		optional	optional	n/a	optional	
Core Standards (CS)					n/a	
Portal						n/a

Table 1: Shared Services Component Startup Dependencies

6.1.1 Shared Services Deployment Order

Clearly from Table 1 above, every shared service is dependent on SSO, and most are dependent on Program Management. These two should be the first components deployed in any system. Following those, it's reasonable to deploy the remaining shared service components shown above. Integration with MNA is optional but strongly encouraged.

6.1.2 SSO Deployment and Provisioning

There are three types of authentication that must be supported by this system.

- *User authentication*, where an application's user interface prompts a user to authenticate via a username and password (SAML).

- *Coordinated web services*, where the application itself makes calls to other applications on behalf of the logged in user, and the remote application must be able to validate the user's credentials in the background (SAML Bearer).
- *Machine to machine authentication*, where an application makes a call to another application on behalf of no user, but must still be validated. The SSO component is responsible for managing all three types of authentication, as well as facilitating authorization, using tokens as appropriate (OAuth).

For additional information on SAML and OAuth, please refer to the documentation in *Shared Security repository* [R15].
For creation and installation information, please refer to the OpenAM [R13] and OpenDJ [R14] repositories, which contain server creation and installation instructions and scripts.
NOTE: It is critical that the system date/time on all SSO servers remain in sync with all SSO-dependent applications. Failure to do so will cause inability to log in, as well as application failures. This is normally done by installing and running an ntp daemon, and is performed automatically by the ansible install described in Section 5 (*Server Creation / Configuration / Provisioning*).

6.1.3 Program Management Deployment

The Program Management component (also referred to as PM or ProgMan) is the master data repository and service. It is a set of services to provide data that crosses component concerns, as well as holding configuration and branding information for other components.

6.1.4 Program Management Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server. The most up-to-date installation instructions are available in the *Program Management source repository* [R2].

6.1.4.1 Property Configuration

The Program Management application has its own set of application-specific configurations to bring it up. Since Program Management is the master data repository which maintains the configurations of all other applications, it is necessary to configure Program Management with the required properties files. These are placed in what is termed the `SB11_CONFIG_DIR`, located under `$CATALINA_HOME/resources`.
A default set of configurations are provided in the code repository. Below is a description of these properties:

Properties file **`progman-bootstrap.properties`** contains all of the following key/value pairs:

MnA properties

- `progman.mna.description` - {a descriptive name for the component used as a display in MnA}
- `mna.mnaUrl` - {url to the base context of the MNA REST application}
- `oauth.access.url` - {url to OAuth URL to OAM instance to allow client calls to POST to get an OAuth token for any 'machine to machine' calls requiring OAUTH}

- `mna.oauth.client.id` - {OAuth Client id configured in OAM to allow get an OAuth token for the ‘batch' web service call to MnA}
- `mna.oauth.client.secret` - {OAuth Client secret/password configured in OAM to allow get an OAuth token for the ‘batch' web service call to core standard}
- `mna.oauth.batch.account=` - {Username (email address) of MNA client user used for authenticating into MNA and logging metrics information}
- `mna.oauth.batch.password=` - {Password of MNA client user}

Mongo Properties

- `pm.mongo.hostname` - {hostname of the mongodb instance}
- `pm.mongo.user` - {mongodb username}
- `pm.mongo.password` - {mongodb password}
- `pm.mongo.dbname` - {mongodb name}

PBE properties

Data encryption requires an externally defined property to hold a secret password.
Password-Based Encryption (PBE) is used to encrypt any sensitive values in the key value configurations. A salt is recommended but not required (the key must exist but the value may be left blank):

- pm.pbe.pass=

In order for encryption of data to work, the unlimited JCE security policy must be installed. Copy `encryption/UnlimitedJCEPolicy/local_policy.jar` and `encryption/UnlimitedJCEPolicy/US_export_policy.jar` into your JDK's lib/security folder, replacing the existing files (please back up existing files). See the README.txt in that directory for more details.

PM config properties

- `pm.rest.service.endpoint` - {fully qualified URL to base context of the rest webservice}
- `pm.rest.context.root` - {relative path to base context of the rest webservice}
- `pm.minJs` - {whether to use minimized javascript in the browser}
- `progman_resource_check_token_url` - {URL of OpenAM OAuth token check}
- `mna.logger.level` - {level of logging that will be sent to the monitoring and alerting. it defaults to ERROR if not set}

PM security properties

- `pm.security.saml.keystore.user` - {the cert for accessing SSO server via https)
- `pm.security.saml.keystore.pass` - {password for cert to access SSO server via https}
- `pm.security.dir` - {file:///opt/... (fully qualified path to location of saml metatdata files)}
- `pm.rest.saml.metadata.filename` - {name of the saml metadata file for the REST app contained in the security.dir configured above}
- `pm.webapp.saml.metadata.filename` - {name of the saml metadata file for the WEBAPP app contained in the security.dir configured above}

- `pm.security.idp` - {fully qualified path to the SAML 2 IDP}

Clustered Environment properties

These are *optional* properties which are used when configuring a clustered environment behind a load balancer (LB). To activate clustered environment support, simply change the active profile setenv as follows: change spring.profiles.active from server.singleinstance to server.loadbalanced. Furthermore, you will need to set these key/value pairs appropriately:

- `progman.loadbalanced.url.scheme` - {this should be http or https}
- `progman.loadbalanced.server.name` - {the loadbalancer's name}
- `progman.loadbalanced.server.port` - {if your server requires a port, include it here, otherwise put 80 in as the default}
- `progman.loadbalanced.includeServerPortInRequestURL` - {boolean true/false value which indicates if the port should be included to resolve the server}
- `progman.loadbalanced.server.rest.contextpath` - {REST context name. e.g.: "/progman.rest"}
- `progman.loadbalanced.server.webapp.contextpath` - {webapp context name. e.g.: "/progman.webapp". Leave this blank if you are using ROOT as webapp context name}

Logback configs

logback configurations can also be placed in this file. These are optional and can be configured in the application's existing logback.xml file instead:

- `logfile.path` - {/path/to/wherever/logs/should/go/; e.g. /usr/local/tomcat/logs}
- `app.base.package.name` - program-management
- `app.context.name` - org.opentestsystem.shared.progman
- `app.base.package.loglevel` - debug

6.1.4.2 Database Configuration

Program Management uses a MongoDB database. Follow the steps in Section 5 of this document to create a generic MongoDB server. Progman will create a database with all required collections upon startup if one doesn't already exist.

6.1.4.3 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values.

```
export CATALINA_OPTS="-DSB11_CONFIG_DIR=$CATALINA_HOME/resources
-DmnaServerName=mna.example.org
-Dspring.profiles.active=mna.client.integration,progman.client.impl.null,special.role
.required
-Djavax.net.ssl.trustStore=/trust/store/certs/path/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXX"
```

`mna.client.integration:` Integrate Progman with MNA. *This should be set to `mna.client.null` if not using MNA or if MNA is not yet available*.
`progman.client.impl.null:` Use the Program Management null implementation because Progman cannot use itself.
`special.role.required:` This allows progman to use a local implementation of a role to permission binding for a given role. This removes the hard, runtime dependency on the Permissions component if it's unavailable.

Please refer to the *Program Management repository* [R2] for more information, especially about permissions and access.

6.1.5 Permissions Deployment

This is a centralized permissions management for the systems components. It is necessary to require that components share the same permissioning capabilities in order to reduce permissions management complexity, and to allow consistent user experience across multiple components developed by different vendors.

6.1.6 Permissions Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server.

6.1.6.1 Property Configuration

Configuration settings for this component are defined via the Program Management application. Add all configuration settings based on the default set provided in the code repository.

Database Properties
The following parameters need to be configured inside program management for database.

- `datasource.url=jdbc:mysql://localhost:3306/schemaname` - The JDBC URL of the database from which Connections can and should be acquired.
- `datasource.username=<db-username>` - Username that will be used for the DataSource's default getConnection() method.
- `encrypt:datasource.password=<db-password>` - Password that will be used for the DataSource's default getConnection() method.
- `datasource.driverClassName=com.mysql.jdbc.Driver` - The fully qualified class name of the JDBC driverClass that is expected to provide Connections.
- `datasource.minPoolSize=5` - Minimum number of Connections a pool will maintain at any given time.
- `datasource.acquireIncrement=5` - Determines how many connections at a time datasource will try to acquire when the pool is exhausted.
- `datasource.maxPoolSize=20` - Maximum number of Connections a pool will maintain at any given time.
- `datasource.checkoutTimeout=60000` - The number of milliseconds a client calling getConnection() will wait for a Connection to be checked in or acquired when the pool is

exhausted. Zero means wait indefinitely. Setting any positive value will cause the getConnection() call to time out and break with an SQLException after the specified number of milliseconds.

- `datasource.maxConnectionAge=0` - Seconds, effectively a time to live. A Connection older than maxConnectionAge will be destroyed and purged from the pool. This differs from maxIdleTime in that it refers to absolute age. Even a Connection which has not been idle will be purged from the pool if it exceeds maxConnectionAge. Zero means no maximum absolute age is enforced.
- `datasource.acquireRetryAttempts=5` - Defines how many times datasource will try to acquire a new Connection from the database before giving up. If this value is less than or equal to zero, datasource will keep trying to fetch a Connection indefinitely.

MNA Properties

The following parameters need to be configured inside Program Management for MNA.

- `mna.mnaUrl=http://<mna-context-url>/mna-rest/` - URL of the Monitoring and Alerting client server's rest url.
- `mnaServerName=permission_dev` - Used by the mna clients to identify which server is sending the log/metrics/alerts.
- `mnaNodeName=dev` - Used by the mna clients to identify who is sending the log/metrics/alerts. There is a discrete mnaServerName and a node for server name & node1/node2 in a clustered environment giving the ability to search across clustered nodes by server name or for a given spcific node. it’s being stored in the DB for metric/log/alert, but is not displayed.
- `mna.logger.level=ERROR` - Used to control what is logged to the Monitoring and Alerting system. Logging levels: ALL (turn on all logging levels), TRACE, DEBUG, INFO, WARN, ERROR, OFF (turn off logging).

SSO Properties

The following parameters need to be configured inside program management for SSO.

- `permission.uri=https://<permission-app-context-url>/rest` - The base URL of the REST api for the Permissions application.
- `component.name=Permissions` - The name of the component that this Permissions deployment represents. This must match the name of the component in Program Management and the name of the component in the Permissions application.
- `permission.security.idp=https://<idp-url>` - The URL of the SAML-based identity provider (OpenAM).
- `permission.webapp.saml.metadata.filename=permissions_local_sp.xml` - OpenAM Metadata file name uploaded for environment and placed inside server directory.
- `permission.security.dir=file:////<sp-file-location-folder>` - Location of the metadata file.
- `permission.security.saml.keystore.cert=<cert-name>` - Name of the Keystore cert being used.
- `permission.security.saml.keystore.pass=<password>` - Password for keystore cert.
- `permission.security.saml.alias=permissions_webapp` - Alias for identifying web application.

- oauth.tsb.client=tsb - OAuth Client id configured in OAM to allow the SAML bearer workflow to convert a SAML assertion into an OAuth token for the ‘coordinated web service” call to TSB.
- oauth.access.url=https://<oauth-url> - OAuth URL to OAM to allow the SAML bearer workflow to POST to get an OAuth token for any ‘machine to machine calls requiring OAuth.
- encrypt:oauth.tsb.client.secret=<password> - OAuth Client secret/password configured in OAM (under the client id) to allow the SAML bearer workflow to convert a SAML assertion into an OAuth token for the "coordinated web service” call to TSB.
- encrypt:mna.oauth.client.secret=<password> - OAuth Client secret/password configured in OAM to allow get an OAuth token for the "batch" web service call to MnA.
- mna.oauth.client.id=mna - OAuth Client ID configured in OAM to allow get an OAuth token for the "batch" web service call to MnA.
- encrypt:permission.oauth.resource.client.secret=<password> - OAuth client secret/password configured in OAM to allow get an OAuth token for the "batch" web service call to permissions.
- permission.oauth.resource.client.id=permissions - OAuth Client ID configured in OAM to allow get an OAuth token for the "batch" web service call to Permissions.
- permission.oauth.checktoken.endpoint=http://<oauth-url> - OAuth URL to OAM to allow the SAML bearer workflow to perform a GET to check that an OAuth token is valid.

6.1.6.2 Database Configuration

Permissions uses a MySQL database. Follow the steps in Section 5 of this document to create a generic MySQL server. Use the SQL scripts provided in the repository to create and populate the Permissions database. InsertStartupData.sql adds the default SBAC hierarchy into the database. A spreadsheet showing the default components, roles, permissions, and mappings of roles to permissions is available at [7].

6.1.6.3 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values. Refer to the *Permissions code repository* [R3] for the latest information.

- `-Dspring.profiles.active` - Active profiles should be comma separated. Typical profiles for this include:
  - `progman.client.impl.integration` - Integrate with Program Management
  - `progman.client.impl.null` - Use the Program Management null implementation
  - `mna.client.integration` - Integrate with MnA component
  - `mna.client.null` - Use the null MnA component
- `-Dprogman.baseUri` - This URI is the base URI where the Program Management REST module is deployed.
- `-Dprogman.locator` - The locator variable describes which combinations of name and environment (with optional overlay) should be loaded from Program Management. For example: "component1-urls,dev" would look up the name component1-urls for the dev environment at the configured REST endpoint. Multiple lookups can be performed by using a semicolon to delimit the pairs (or triplets with overlay): "component1-urls,dev;component1-other,dev"
- `-DSB11_CONFIG_DIR` - Locator string needed to find the Permissions properties to load.

- `-Djavax.net.ssl.trustStore` - Location of .jks file which contains security certificates for SSO, Program Management and Permission URLs specified inside baseUri and Program Management.
- `-Djavax.net.ssl.trustStorePassword` - Password string for the keystore.jks file.

Example:

```
export CATALINA_OPTS=" -Xms512m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m
-Dspring.profiles.active=progman.client.impl.integration,mna.client.integration
-Dprogman.baseUri=http://pm.example.org:8080/rest/
-Dprogman.locator="permissions,Dev" -DmnaServerName=perm.example.org
-Djavax.net.ssl.trustStore=/trust/store/certs/path/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXX"
```

Please refer to the *Permissions code repository* [R3] for more information.

6.1.7 Monitoring and Alerting Deployment

Monitoring and Alerting, also known as MnA, is a shared set of services that allow components to send alerts in a consistent way. Also those alerts can be monitored and acted upon in a similarly consistent way. It also will allow vendors to develop add on applications and features to use and act on these alerts.

6.1.8 Monitoring and Alerting Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server.

6.1.8.1 Property Configuration

Configuration settings for this component are defined via the Program Management application. Add all configuration settings based on the default set provided in the code repository [R4]. Please refer there for the latest information on setup and configuration.

- `mna.mongo.hostname=mongo.host` - The server on which MongoDB is running
- `mna.mongo.port=27017` - MongoDB port
- `mna.mongo.user=` - MongoDB user
- `mna.mongo.password=` MongoDB password
- `mna.mongo.dbname=mna` - Name of the MongoDB database
- `mna.email.active=true` - Enable or disable email from MNA
- `mna.email.address.from=from@somecorp.com` - From address when email from MNA is sent
- `mna.email.subject.prefix=TEST EMAIL ONLY -` - Subject of email from MNA
- `mna.email.host=email.host` - Host address of mail server
- `mna.email.port=465` - Email port
- `mna.email.user=emailuser` - Email username
- `mna.email.password=emailpassword` - Email password
- `mna.email.smtp.starttls.enable=true` - Start TLS enabled
- `mna.email.transport.protocol=smtp` - Email protocol used
- `mna.email.smtp.auth=true` - Use email authentication?
- `mna.email.smtp.ssl.enable=true` - Enable ssl over SMTP

- mna.rest.context.root=/rest/ - MNA REST context root path
- mna.clean.days=30 - How long to retain logs before cleaning out. Not required. defaults to 30
- mna.clean.cron=0 0 0 * * ? - timing for cron job. Not required. defaults to 0 0 0 * * ?

Clustered Environment properties

These are *optional* properties which are used when configuring a clustered environment behind a load balancer (LB). To activate clustered environment support, simply change the active profile setenv as follows: change `spring.profiles.active` from `server.singleinstance` to `server.loadbalanced`. Furthermore, you will need to set these key/value pairs appropriately:

- `mna.loadbalanced.url.scheme` - {this should be http or https}
- `mna.loadbalanced.server.name` - {the loadbalancer's name}
- `mna.loadbalanced.server.port` - {if your server requires a port, include it here, otherwise put 80 in as the default}
- `mna.loadbalanced.includeServerPortInRequestURL` - {boolean true/false value which indicates if the port should be included to resolve the server}
- `mna.loadbalanced.server.rest.contextpath` - {REST context name. e.g.: "/mna.rest"}
- `mna.loadbalanced.server.webapp.contextpath` - {webapp context name. e.g.: "/mna.webapp". Leave this blank if you are using ROOT as webapp context name}

6.1.8.2 Database Configuration

Monitoring and Alerting uses a MongoDB database. Follow the steps in Section 5 of this document to create a generic MongoDB server. MNA will create a database with all required collections upon startup if one doesn't already exist.

6.1.8.3 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values.

```
export CATALINA_OPTS="-Djava.rmi.server.hostname=mna.example.org
-DSB11_CONFIG_DIR=$CATALINA_HOME/resources
-DmnaServerName=mnaexample.org
-Dspring.profiles.active=progman.client.impl.integration
-Dprogman.baseUri=https://pm.example.org/rest/
-Dprogman.locator=MNA,web-dev
-Djavax.net.ssl.trustStore=/trust/store/certs/path/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXX"
```

Please refer to the *Monitoring and Alerting repository* [R4] for more information.

6.1.9 Core Standards Deployment

This is the component that needs to manage the Common Core State Standards and learning metadata so that other components can reference and use them in the same manner. It is the

single version of truth for these standards.The standards publication must be stored in an OpenOffice Calc spreadsheet with a .ods extension. OpenOffice was selected because it is an open-source spreadsheet application in keeping with the open-source nature of the technologies used in the Smarter Balanced Assessment System.

6.1.10 Core Standards Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server.

6.1.10.1 Property Configuration

Configuration settings for this component are defined via the Program Management application. Modify the configuration settings based on the default set provided in the code repository.

Database Properties

The following parameters need to be configured inside Program Management for database.

- `datasource.url=jdbc:mysql://<db.url>:<db.port>/<schemaname>` - The JDBC URL of the database from which Connections can and should be acquired. Can be localhost. Port is usually 3306.
- `datasource.username=<db-username>` - Username that will be used for the DataSource's default getConnection() method.
- `datasource.password=<db-password>` - Password that will be used for the DataSource's default getConnection() method.
- `datasource.driverClassName=com.mysql.jdbc.Driver` - The fully qualified class name of the JDBC driverClass that is expected to provide Connections.
- `datasource.minPoolSize=5` - Minimum number of Connections a pool will maintain at any given time.
- `datasource.acquireIncrement=5` - Determines how many connections at a time datasource will try to acquire when the pool is exhausted.
- `datasource.maxPoolSize=20` - Maximum number of Connections a pool will maintain at any given time.
- `datasource.checkoutTimeout=60000` - The number of milliseconds a client calling getConnection() will wait for a Connection to be checked in or acquired when the pool is exhausted. Zero means wait indefinitely. Setting any positive value will cause the getConnection() call to time out and break with an SQLException after the specified number of milliseconds.
- `datasource.maxConnectionAge=0` - Seconds, effectively a time to live. A Connection older than maxConnectionAge will be destroyed and purged from the pool. This differs from maxIdleTime in that it refers to absolute age. Even a Connection which has not been idle will be purged from the pool if it exceeds maxConnectionAge. Zero means no maximum absolute age is enforced.

- `datasource.acquireRetryAttempts=5` - Defines how many times datasource will try to acquire a new Connection from the database before giving up. If this value is less than or equal to zero, datasource will keep trying to fetch a Connection indefinitely.

MNA (Monitoring and Alerting) properties

The following parameters need to be configured inside program management for MNA.

- `mna.mnaUrl=http://<mna-context-url>/mna-rest/` - URL of the Monitoring and Alerting client server's rest url.
- `mnaServerName=corestandards_dev` - Used by the MNA clients to identify which server is sending the log/metrics/alerts.
- `mnaNodeName=dev` - Used by the MNA clients to identify who is sending the log/metrics/alerts. There is a discrete mnaServerName and a node in case you want to say XXX for server name & node1/node2 in a clustered environment giving you the ability to search across clustered nodes by server name or specifically for a given node. It's being stored in the db for metric/log/alert, but not displayed.
- `mna.logger.level=ERROR` - Used to control what is logged to the Monitoring and Alerting system. Logging Levels (ALL - Turn on all logging levels; TRACE, DEBUG, INFO, WARN, ERROR, OFF - Turn off logging).

SSO properties

The following parameters need to be configured inside Program Management for SSO.

- `permission.uri=https://<permission-app-context-url>/rest` - The base URL of the REST API for the Permissions application.
- `component.name=CoreStandards` - The name of the component that this CoreStandards deployment represents. This must match the name of the component in Program Management and the name of the component in the Permissions application.
- `corestandards.security.idp=https://<idp-url>` - The URL of the SAML-based identity provider (OpenAM).
- `corestandards.webapp.saml.metadata.filename=corestandards_local_sp.xml` - Name of OpenAM SP (Service Provider) Metadata file which has been uploaded for the environment, as well as placed inside the server's filesystem.
- `corestandards.security.dir=file:////<sp-file-location-folder>` - Location of the metadata file.
- `corestandards.security.saml.keystore.cert=<cert-name>` - Name of the Keystore cert being used.
- `corestandards.security.saml.keystore.pass=<password>` - Password for keystore cert.
- `corestandards.security.saml.alias=corestandards_webapp` - Alias for identifying the web application.
- `oauth.tsb.client=tsb` - OAuth Client id configured in OAM to allow the SAML bearer workflow to convert a SAML assertion into an OAuth token for the 'coordinated web service" call to TSB.
- `oauth.access.url=https://<oauth-url>` - OAuth URL to OAM to allow the SAML bearer workflow to POST to get an OAuth token for any "machine to machine" calls requiring OAUTH.

- encrypt:oauth.tsb.client.secret=<password> - OAuth Client secret/password configured in OAM (under the client id) to allow the SAML bearer workflow to convert a SAML assertion into an OAuth token for the "coordinated web service" call to TSB.
- encrypt:mna.oauth.client.secret=<password> - OAuth Client secret/password configured in OAM to allow get an OAuth token for the "batch" web service call to MnA.
- mna.oauth.client.id=mna - OAuth Client id configured in OAM to allow get an OAuth token for the "batch" web service call to MnA.
- encrypt:corestandards.oauth.resource.client.secret=<password> - OAuth Client secret/password configured in OAM to allow it to get an OAuth token for the "batch" web service call to Core Standards.
- corestandards.oauth.resource.client.id=corestandards - OAuth Client id configured in OAM to allow it to get an OAuth token for the "batch" web service call to Core Standards.
- corestandards.oauth.checktoken.endpoint=http://<oauth-url> - OAuth URL to OAM to allow the SAML bearer workflow to perform a GET to check that an OAuth token is valid.

6.1.10.2 Database Configuration

Core Standards uses a MySQL database. Follow the steps in Section 5 of this document to create a generic MySQL server. Use the SQL script provided in the repository to create and populate the Core Standards database. Before executing the scripts, create the database by executing the following command:

```
create schema StandardsRepository_Dev default character set=UTF8;
```

Following this, execute the script `Create_StandardsRepository_Tables.sql`, followed by `StandardsRepository_sp_and_functions.sql`. Finally, to insert sample data into the DB, execute the `Sample_Data_for_StandardsRepository_db.sql` script.

6.1.10.3 Open Office (OO) configuration

***NOTE**: This step is automatically performed by the transform_server.sh script discussed above. The provisioning script also performs this installation if necessary.*

After completion of the Open Office installation, create a symlink called */usr/bin/soffice* pointing to */install/directory/path/openoffice4/program/soffice.* Finally add OO resource information to tomcat context.xml file.

```
<Parameter name="ooExecPath" value="/install/directory/path/openoffice4/program"
 type="java.lang.String" override="false" />
```

6.1.10.4 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values.

```
export CATALINA_OPTS="-Xms512m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m
-Djavax.net.ssl.trustStore=/trust/store/certs/path/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXXX"
```

Please refer to the *Core Standards source repository* [R5] for more information.

6.1.11 Portal Deployment

This is the entry point where end-users access the components of the Smarter Balanced system. Through its integration with SSO, it provides the logged-in user access to only the components he/she has access to. Portal is a Wordpress-based application and as such its setup differs from most other SmarterApp components.
Please refer to the Portal source code repository [R12] for the latest installation details and documentation.

6.1.12 Portal Provisioning

- Install Wordpress on a Linux box (Ubuntu Linux 12.04+ recommended)
- Configure Wordpress according to the instructions provided in the Documentation folder.
- Log in to the WP admin interface as WP Admin and install the SAML 2.0 plugin code from this Portal repository, not the official saml-20-single-sign-on plugin.

6.1.12.1 Property Configuration

Configuring the Permissions application

- Create an entry for every component under "Manage Components."
- Portal makes API calls to both *Program Management [R2]* and *Permissions [R3]*. Program Management consumes roles directly, and they are aligned with specific permissions (refer to [R2]). The role required for obtaining Program Management information is **Program Management Read**, which is aligned with the **Progman Read** permission.
- There is no need to assign new permissions to the Portal components.

Creating the Portal User in ART

- Create a new user in ART [R9] for Portal's backend processing. The user must have the **Program Management Read** role, assigned at the highest available hierarchy level.

Configuring the Program Management application

- Create an entry for each component being served by Portal. The "component names" in Program Management must exactly match the component names in Permissions application.
- Create a configuration for Portal containing a set of properties. Program Management properties need to be set for proper functioning of the logged-in portion of the Portal. Sample Portal properties may be found at [config/portal-progman-config.txt]. What follows is an explanation of each Program Management key/value pair.

`iconRelativePath=wp-content/themes/smarterbalanced/images/component_icons/` - Local WP relative path of SmarterApp icon location for display on the logged-in Portal page.

For each component defined in the SmarterApp ecosystem, the following four key/value pairs must be defined:

```
COMPONENT NAME=component
COMPONENTNAME.displayname=COMPONENT NAME
COMPONENTNAME.url=https://COMPONENT.URL
COMPONENTNAME.icon=COMPONENT.png
```

As a working example, this is how the Core Standards component would be set up.

- `Core Standards=component` - Defines a component named "Core Standards." This component name, stripped of any spaces, becomes the identifier for the remaining three keys.
- `CoreStandards.displayname=Core Standards` - Defines the name to use in Portal when displaying the icon for this component. The display name does not have to exactly match the component name defined above.
- `CoreStandards.url=https://cs.url/` - Defines the URL for the component itself; when logged-in users click here, they are redirected to this URL.
- `CoreStandards.icon=core_standards.png` - Defines the icon name for this component. The icon must be located on the local file system, at the location defined in `iconRelativePath`.
- Repeat the above for each component that needs to be accessible from Portal.

Now Configure SSO/SAML on the Portal server

- Go to `/var/www/wordpress/wp-content/plugins` and rename `saml-20-single-sign-on` to, say, `saml-20-single-sign-on-`
- Log into the portal as WP Admin
- Go to `/var/www/wordpress/wp-content/plugins and rename saml-20-single-sign-on-` back to `saml-20-single-sign-on`
- In the WP admin console, go to Settings -> Single Sign-On, and navigate to the Identity Provider tab.
- In the first URL box (IdP URL) enter the IdP config URL (e.g. https://OAM.URL/auth/saml2/jsp/exportmetadata.jsp?realm=/your_realm) and click "Fetch Metadata," then Update Options if all looks correct.
- Log in to the associated OpenAM instance, go to Federation tab, and add Entity. Enter the WP SP URL (e.g. `http://portal.your.portal.domain/wp-content/plugins/saml-20-single-sign-on/saml/www/module.php/saml/sp/metadata.php/1`), and save.
- Click on the realm (e.g., sbac) and add the newly added SP to the Circle of Trust.
- Provision a new user in your SSO system using the Administration and Registration Tools (ART) component named portal.agent@example.com, with a Role of Portal Agent at the Client level.

Configuring the Portal Backend Service

Several services need to run on Portal in order to integrate with SSO, Program Management, and Permissions.

- Copy `dump_component_mapping.pl` to `/usr/local/bin/dump_component_mapping.pl`
- Copy `dump_component_mapping.sh` to `/usr/local/bin/dump_component_mapping.sh`

- Install a crontab for root to execute `/usr/local/bin/dump_component_mapping.sh` periodically - for example, once or twice daily.

6.2 Assessment Creation and Management Components

These components manage the process and workflow of the creation and life cycle of Assessment Artifacts. See the chart below for the components associated with this subsection. Note that *Test Packager* functionality has been combined with the *Test Authoring* component. See [4] for more information on dependencies.

Deployed Component → / Dependent component ↓	Test Authoring	Test Spec Bank	Test Item Bank
Single Sign On (SSO)	Y	Y	
Program Management (PM)	Y	Y	Y
Permissions	Y	Y	
Monitoring & Alerting (MNA)	optional	optional	optional
Core Standards (CS)	Y		
Test Authoring	n/a	Y	
Test Spec Bank (TSB)	Y	n/a	
Test Item Bank (TIB)	Y	Y	n/a
FTP server		Y	Y
Database server	MongoDB	MongoDB	MongoDB

Table 2: Assessment Creation and Management Component Dependencies

6.2.1 Test Authoring Deployment

This component is a graphical interface used for creating test blueprints and specifications and manage the workflow. It interacts with the Test Item Bank and the Test Spec Bank components. This component includes Test Packaging functionality, which prepares the test items and the test specifications for use by the test delivery system. Test packager pre-processes assessment assets to make them more efficient for the Test Delivery component. The packager creates the assessment instrument that a Test Delivery system can consume and use to deliver the assessment.

6.2.2 Test Authoring Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server.

6.2.2.1 Property Configuration

Configuration settings for this component are defined via the Program Management application. Add all configuration settings based on the default set provided in the code repository [R6]. Please refer there for the latest information on setup and configuration.

- `testauth.item.count.max.limit=40000` - Maximum items per assessment
- `component.name=TestAuthoring` - Component name must match name in Permissions and Program Management
- `permission.uri=http://name.of.permissions.server/rest` - URI of the Permissions application's REST endpoint
- `testauth.security.idp=http://name.of.identity.provider/auth/saml2/jsp/exportmetadata.jsp?metalias=idp&realm=sbac` - SSO Identity Provider metadata URL
- `testauth.mna.description=The Test Authoring Component` - Name of this component, as shown within Monitoring and Alerting
- `testauth.mna.healthMetricIntervalInSeconds=120` - Periodic health metrics for MNA
- `mna.mnaUrl=http://name.of.mna.server/rest/` - URL of MNA REST endpoint
- `mna.logger.level=[OFF | ERROR | WARN | INFO | DEBUG | TRACE | ALL](default:ERROR)` - MNA logging level
- `mna.clean.days=30 (default)` - How long to keep logs before cleanup
- `mna.clean.cron=0 0 0 * * ? (default)` - MNA cron job for cleanup
- `mna.oauth.batch.account=` - Username (email address) of MNA client user used for authenticating into MNA and logging metrics information
- `mna.oauth.batch.password=` - Password of MNA client user
- `oauth.tsb.client=` - Oauth client id
- `oauth.tsb.client.secret=` - Oauth TSB client secret
- `oauth.tsb.batch.account=` - Account name for machine user being used to publish to TSB
- `oauth.tsb.batch.password=` - TSB machine user password
- `testauth.mna.availability.metric.email=abc@example.org` - email to send availability metrics
- `testauth.mongo.hostname=` - Mongo DB Host name
- `testauth.mongo.port=27017` - Mongo DB port
- `testauth.mongo.username=` - Mongo DB usename
- `testauth.mongo.password=` - Mongo DB password
- `testauth.mongo.dbname=testauth-dev` - Mongo DB database name
- `testauth.minJs=true` - Use minJs?
- `testauth.rest.context.root=/rest/` - REST endpoint of Test Authoring
- `testauth.dtd.url=http://xxxxx/rest/resources/dtd/testpackage_v_9_19_2013.dtd` - Test Package DTD path
- `tsb.tsbUrl=http://name.of.test.spec.bank.server/rest/` - Test Spec Bank REST endpoint URL
- `testauth.core.standards.url=http://name.of.corestandards.server/api/` - CoreStandards API endpoint URL

- `tib.baseUri=http://name.of.test.item.bank.server/rest/` - Test Item Bank REST endpoint URL
- `testauth.security.profile=dev` -
- `testauth.languages=Afar|aar|Abkhazian|abk|Achinese|ace...` - Pipe-delimited list of supported languages in Test Authoring, e.g. Name|three-letter-official-abbreviation.

Clustered Environment Properties

These are *optional* properties which are used when configuring a clustered environment behind a load balancer (LB). To activate clustered environment support, simply change the active profile setenv as follows: change `spring.profiles.active` from `server.singleinstance` to `server.loadbalanced`. Furthermore, you will need to set these key/value pairs appropriately:

- `testauth.loadbalanced.url.scheme` - {this should be http or https}
- `testauth.loadbalanced.server.name` - {the loadbalancer's name}
- `testauth.loadbalanced.server.port` - {if your server requires a port, include it here, otherwise put 80 in as the default}
- `testauth.loadbalanced.includeServerPortInRequestURL` - {boolean true/false value which indicates if the port should be included to resolve the server}
- `testauth.loadbalanced.server.rest.contextpath` - {REST context name. e.g.: "/testauth.rest"}
- `testauth.loadbalanced.server.webapp.contextpath` - {webapp context name. e.g.: "/testauth.webapp". Leave this blank if you are using ROOT as webapp context name}

6.2.2.2 Database Configuration

Test Authoring uses a MongoDB database. Follow the steps in Section 5 of this document to create a generic MongoDB server. Test Authoring will create a database with all required collections upon startup if one doesn't already exist.

6.2.2.3 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values.

```
export CATALINA_OPTS="-Xms512m -Xmx1024m -XX:PermSize=512m
-XX:MaxPermSize=512m
-Dspring.profiles.active=progman.client.impl.integration,mna.client.integratio
n -Dprogman.baseUri=https://pm.example.org/rest/
-Dprogman.locator='name,environment' -DmnaServerName=auth.example.org
-Djavax.net.ssl.trustStore=/trust/store/certs/path/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXXX"
```

Please refer to the *Test Authoring repository* [R6] for more information.

6.2.3 Test Spec Bank Deployment

Test Specification Bank (a.k.a. TSB or Test Spec Bank) is a repository for test specifications, blueprints and other data about tests such as the adaptive algorithm to be used during the test.

6.2.4 Test Spec Bank Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server.

6.2.4.1 Property Configuration

Configuration settings for this component are defined via the Program Management application. Add all configuration settings based on the default set provided in the code repository [R7]. Please refer there for the latest information on setup and configuration.

- `tsb.security.idp=http://name.of.identity.provider.server/auth/saml2/jsp/exportmetadata.jsp?metalias=idp&realm=sbac` - URL of SSO IDP's metadata export file
- `permission.uri=http://name.of.permissions.server/rest` - URI to Permissions REST interface
- `component.name=TestSpecBank` - Must match the name in Permissions and Program Management applications
- `tsb.mna.description=The Test Spec Bank Component` - TSB description for MNA logs
- `mna.mnaUrl=http://name.of.mna.server/rest` - URL to MNA's REST interface
- `mna.logger.level=[OFF | ERROR | WARN | INFO | DEBUG | TRACE | ALL] (default:ERROR)` - Error log items of this level or higher are to be sent to the MNA application.
- `mna.clean.days=30 (default)` - Clear our MNA logs after this many days
- `mna.clean.cron=0 0 0 * * ? (default)` - Cron job entry for MNA log cleaning schedule
- `mna.oauth.batch.account` - Username (email address) of MNA client user used for authenticating into MNA and logging metrics information
- `mna.oauth.batch.password=` - Password of MNA client user
- `tsb.mongo.hostname=` - TSB Mongo DB hostname
- `tsb.mongo.port`=27017 - Mongo DB port
- `tsb.mongo.username=` - Mongo DB username
- `tsb.mongo.password=` - Mongo DB password
- `tsb.mongo.dbname=tsb-dev` - Mongo DB database name
- `tsb.dtd.url=http://name.of.test.authoring.server/rest/resources/dtd/testpackage_v_9_19_2013.dtd` - Test package DTD location
- `tsb.rest.context.root=/rest/` - Root context of the TSB REST service
- `tsb.minJs=true` - Whether to use minimized JavaScript in the browser, true to minify
- `tib.tibUrl=http://name.of.test.item.bank.server/` - URL to TIB server
- `tsb.sftp.host=` - TSB SFTP host
- `tsb.sftp.port=22` - TSB SFTP port
- `tsb.sftp.user=` - TSB SFTP account username
- `tsb.sftp.pass=` - TSB SFTP account password
- `tsb.sftp.dir=` - TSB SFTP account directory
- `tsb.tib.sftp.host=` - TSB's TIB SFTP hostname
- `tsb.tib.sftp.port=22` - TSB's TIB SFTP server port
- `tsb.tib.sftp.user=` - TSB's TIB SFTP account
- `tsb.tib.sftp.pass=` - TSB's TIB SFTP account password

- `tsb.download.directory=` - TSB's SFTP download directory
- `tsb.export.cron.trigger=0,30 * * * * ?` - Cron to configure export frequency of test specifications

6.2.4.2 Clustered Environment properties

These are *optional* properties which are used when configuring a clustered environment behind a load balancer (LB). To activate clustered environment support, simply change the active profile setenv as follows: change `spring.profiles.active` from `server.singleinstance` to `server.loadbalanced`. Furthermore, you will need to set these key/value pairs appropriately:

- `tsb.loadbalanced.url.scheme` - {this should be http or https}
- `tsb.loadbalanced.server.name` - {the loadbalancer's name}
- `tsb.loadbalanced.server.port` - {if your server requires a port, include it here, otherwise put 80 in as the default}
- `tsb.loadbalanced.includeServerPortInRequestURL` - {boolean true/false value which indicates if the port should be included to resolve the server}
- `tsb.loadbalanced.server.rest.contextpath` - {REST context name. e.g.: "/tsb.rest"}
- `tsb.loadbalanced.server.webapp.contextpath` - {webapp context name. e.g.: "/tsb.webapp". Leave this blank if you are using ROOT as webapp context name}

6.2.4.3 Database Configuration

Test Spec Bank uses a MongoDB database. Follow the steps in Section 5 of this document to create a generic MongoDB server. Test Authoring will create a database with all required collections upon startup if one doesn't already exist.

6.2.4.4 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values.

```
export CATALINA_OPTS=" -Xms512m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m
-Dspring.profiles.active=progman.client.impl.integration,mna.client.integration
-Dprogman.baseUri=https://pm.example.org/rest/
-Dprogman.locator=Name,Environment -DmnaServerName=tsb.example.org
-Djavax.net.ssl.trustStore=/trust/store/certs/path/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXX"
```

6.2.4.5 Side-Loading External Packages

To load externally authored test packages into the TSB, you may use the "load_reg_package.pl" script available in the `tsb` folder of the Administrative repository [R1]. Test packages will then be browsable in ART's *Select Assessment* function. Refer to the script's usage instructions for more details on script usage.

Please refer to the *Test Spec Bank repository* [R7] for more information.

6.2.5 Test Item Bank Deployment

This component contains items that are in operational, field, or interim tests.

6.2.6 Test Item Bank Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server.

6.2.6.1 Property Configuration

Configuration settings for this component are defined via the Program Management application. Add all configuration settings based on the default set provided in the code repository [R8]. Please refer there for the latest information on setup and configuration.

- `component.name=Test Item Bank` - Component name. Must match name in Permissions and Program Management.
- `mna.mnaUrl=http://name.of.mna.server/rest/` - URL of the MNA server's REST endpoint
- `mna.oauth.client.id=mna` - OAuth MNA client name
- `mna.oauth.client.secret=` - OAuth MNA client secret
- `mna.oauth.batch.account=` - Username (email address) of MNA client user used for authenticating into MNA and logging metrics information
- `mna.oauth.batch.password=` - Password of MNA client user
- `oauth.access.url=https://name.of.sso.server/auth/oauth2/access_token?realm=/sbac` - SSO OAuth service URL including realm
- `permission.uri=https://name.of.permissions.server/rest` - REST endpoint of Permissions server
- `tib.clean.files.age.days=100` - How many days to keep files in TIB's SFTP directory on the sftp server
- `tib.clean.files.cron.trigger=0 0 8 * * *` - Cron config for cleaning TIB files
- `tib.file.pathname=/usr/local/tomcat/uploads` - Location on the TIB server where TIB will place test items it FTPs from the FTP server.
- `tib.mna.description=The Test Item Bank Component` - Name of TIB component for display in MNA
- `tib.mongo.dbname=test` - TIB Mongo DB database name
- `tib.mongo.hostname=dns.name.of.mongodb.host` - DNS name of the Mongo DB host for TIB.
- `tib.mongo.password=` - TIB Mongo DB password
- `tib.mongo.port=27017` - TIB Mongo DB server port
- `tib.mongo.username=` - TIB Mongo DB username
- `tib.oauth.checktoken.endpoint=https://name.of.sso.server/auth/oauth2/tokeninfo?realm=/sbac` - SSO OAuth checktoken service URL including realm
- `tib.oauth.resource.client.id=` - OAuth TIB client ID
- `tib.oauth.resource.client.secret=` - OAuth TIB client secret
- `tib.sftp.hostname=name.of.sftp.server` - Name of SFTP server from which TIB reads items.
- `tib.sftp.import.directory=` - Import directory name in SFTP directory for TIB user
- `tib.sftp.password=` - TIB SFTP password
- `tib.sftp.port=22` - TIB SFTP port
- `tib.sftp.user=` - TIB SFTP username

6.2.6.2 Database Configuration

Test Item Bank (also known as TIB) uses a MongoDB database. Follow the steps in Section 5 of this document to create a generic MongoDB server. TIB will create a database with all required collections upon startup if one doesn't already exist.

6.2.6.3 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values.

```
export CATALINA_OPTS=" -Xms512m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m
-Dspring.profiles.active=progman.client.impl.integration,mna.client.integration
-Dprogman.baseUri=https://pm.example.org/rest/
-Dprogman.locator=Name,Environment -DmnaServerName=tib.example.org
-Djavax.net.ssl.trustStore=/trust/store/certs/path/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXXX"
```

Please refer to the *Test Item Bank repository* [R8] for more information.

6.3 Assessment Delivery Components

These components deliver the assessment to the students and gather the data and metadata about the assessment. See the chart below for the components associated with this subsection, and [4] for more details.

Deployed Component → / Dependent component ↓	Admin and Registration Tool	TDS / Student	TDS / Proctor	Item Scoring
Single Sign On (SSO)	Y		Y	
Program Management (PM)	Y		Y	
Permissions	Y		Y	
Monitoring & Alerting (MNA)	optional	optional	Y	
Admin and Registration Tool	n/a	Y	Y	
TDS / Student		n/a		
TDS / Proctor			n/a	
Item Scoring				n/a
Data Warehouse				

FTP server	Y			
Database server	MongoDB	MySQL	MySQL	MySQL

Table 3: Assessment Delivery Component Dependencies

6.3.1 Admin and Registration Tools (ART) Deployment

This component manages the capabilities and methods required for assessment scheduling, test windowing, room scheduling, proctor assignment, student assignment, and student identification methods. This component also registers the student(s) for assessments and allows for uploads of student information and their accessibility profile. It must also manage staff identification for managing assessment events.

6.3.2 Admin and Registration Tools (ART) Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server.

6.3.2.1 Property Configuration

Configuration settings for this component are defined via the Program Management application. Add all configuration settings based on the default set provided in the code repository [R9]. Please refer there for the latest information on setup and configuration.

User Change / SSO SFTP Setup

The SSO integration periodically checks to see if any user changes have occurred. If so, it will generate an XML file which is then transferred via SFTP to an OpenAM server where it is consumed for user provisioning or maintenance.

There are a few properties that must be set up in Program Management to enable this integration.

- testreg.user.export.frequency.milliseconds - Periodicity of XML generation in milliseconds
- testreg.sftp.port - SFTP port of server to connect to
- testreg.sftp.dir - Directory on SFTP server to copy file to
- testreg.sftp.pass - Password for authentication
- testreg.sftp.user - User for authentication
- testreg.sftp.host - SFTP host to connect to
- testreg.sso.filename.suffix - Suffix to append to file when SFTP'd

Data Warehouse Integration Setup

The data warehouse integration is a periodic export of data to a data warehouse landing zone. It incorporates pulling data out of the ART database, compressing it, encrypting it, and then SFTPing the data. There are numerous items to setup in order for it to all work.

GPG Keys

All files sent to the data warehouse must be encrypted and signed using GPG. In order to do this, each ART deployment must generate its own GPG key pair.

GPG is usually pre-installed on most UNIX systems. If generating keys on a Mac system install GPG Suite [https://gpgtools.org/]. If generating keys on a Windows system GPG4Win [http://www.gpg4win.org/] can be used.

Follow the directions in the software on your system to generate a key pair. In order to generate a key pair on a UNIX system the following can be used:

```
gpg --gen-key
```

You will be asked to choose a type:

```
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
  Your selection?
```

The default key will work fine.

The next question that is asked is to choose key size. The default key size is ok.

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
```

Now choose how long the key is valid for. For testing purposes it is ok to choose that it will never expire.

```
Requested keysize is 2048 bits
Please specify how long the key should be valid.
    0 = key does not expire
    <n>  = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
 Key is valid for? (0)
```

A name and email address is then requested in order to build a user id to identify the key. For testing purposes these can be fake values.

You need a user ID to identify your key; the software constructs the user ID from the Real Name, Comment and Email Address in this form: "Heinrich Heine (Der Dichter) heinrichh@duesseldorf.de"

```
Real name: Test Tester
Email address: test@test.com
Comment: This is a test account
You selected this USER-ID:
```

```
    "Test Tester (This is a test account) <test@test.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit?
```

Choose (O)kay to continue.

Next choose a passphrase to protect your secret key.

```
You need a Passphrase to protect your secret key.

Enter passphrase:
```

After this, GPG needs to gather entropy in order to generate a more random key. On a system with a GUI and mouse entropy is easy to generate just by moving the mouse around the screen. If logged in remotely to a UNIX system, generating entropy will be a little bit tougher. One way to generate the needed entropy is to open up another SSH session to the remote server and running one or both of the following:

```
ls -R /

find / -type f
```

After the key generates GPG will output something like this:

```
 gpg: key 00168441 marked as ultimately trusted
  public and secret key created and signed.

  gpg: checking the trustdb
  gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
  gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
  pub   2048R/00168441 2014-03-05
        Key fingerprint = 573D 9B85 618A 1FA8 C61A  B07A 4D57 69AE 0016 8441
     uid                  Test Tester (This is a test account) <test@test.com>
     sub   2048R/F288019A 2014-03-05
```

The key pair is now created.

The public key that was created needs to be exported so that it can be given to the data warehouse. The export can be done by:

```
gpg --export -a "test@test.com" > public.key
```

The exported key is an ASCII file.

The Data Warehouse should send a public key to ART. This key should be imported into the keychain on the ART server.

```
gpg --import dw.public.key
```

SSH Key

An SSH key pair will need to be generated for SFTP authentication. It is easiest to generate the SSH key on the server that the SFTP will take place on.

To generate a key pair log in to the remote server and change the user to the correct account that will be SFTPing.

Now use:

```
ssh-keygen
```

SSH-keygen will ask for a file to save the key in.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
```

The default is usually OK unless the user account has multiple ssh keys.

Now ssh-keygen asks for a passphrase. Usually a passphrase is strongly suggested. However, in the case that the application is performing a system to system automated file transfer it is probably ok to just press enter to not have a passphrase.

```
Enter passphrase (empty for no passphrase):
```

You'll see some information about the key and it's now been generated.

```
Your identification has been saved in /home/ubuntu/.ssh/id_rsa.
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub.
The key fingerprint is:
1a:3c:74:09:65:39:8a:b1:c2:48:2e:a3:6b:8d:80:aa ubuntu@jenkinsmaster01
The key's randomart image is:
+--[ RSA 2048]----+
|      ..o.       |
| .  .  oo.       |
|oo   +..o.       |
|+.o oo..         |
|+. .  + S        |
|+      +         |
|o.o   .          |
|oo .             |
|E                |
+-----------------+
```

The public key that was generated needs to be given to the data warehouse so that they can correctly setup security for the landing zone that ART will SFTP to. The path of the generated public key is displayed above.

Password File

A property file needs to be created that contains the passphrase for the secret GPG key that was generated above. This file should contain one line like:

```
testreg.secret.passphrase=testkey
```

The value of that passphrase should be whatever was typed in when the GPG key was generated. Place this file somewhere in the file system that is secure. The only system user that should have access to this file is the user that runs Tomcat.

Tomcat Parameter

Tomcat needs to know where the secret passphrase file is. In order to tell Tomcat about it, a -D parameter needs to be added to the Tomcat command line at startup. It should look something like:

```
-Dtestreg.secret.passphrase.file=/pathto/testregkeypass.properties
```

The parameter name must be testreg.secret.passphrase.file.

Program Management Properties

There are number of other properties that need to be added to Program Management in order to successfully run the data warehouse integration.

- gpgKeyring.public.location - location of GPG public keyring, usually found in .gnupg/pubring.gpg
- gpgKeyring.secret.location - location of GPG secret keyring, usually found in .gnupg/secring.gpg
- testreg.secret.key.userid - email address for test reg key
- landingzone.public.key.userid - email address for landing zone key
- dw.export.callback.url - callback URL for Data Warehouse
- dw.host - SFTP host for landing zone
- dw.private.key.loc - file path of private key for authenticating to landing zone
- dw.port - SFTP port
- dw.user - user to connect to landing zone as
- dw.remote.dir - remote SFTP directory to write to
- dw.gpgfile.prefix - file name prefix for generated file

Clustered Environment properties

These are *optional* properties which are used when configuring a clustered environment behind a load balancer (LB). To activate clustered environment support, simply change the active profile setenv as follows: change `spring.profiles.active` from `server.singleinstance` to `server.loadbalanced`. Furthermore, you will need to set these key/value pairs appropriately:

- `testreg.loadbalanced.url.scheme` - {this should be http or https}
- `testreg.loadbalanced.server.name` - {the loadbalancer's name}
- `testreg.loadbalanced.server.port` - {if your server requires a port, include it here, otherwise put 80 in as the default}
- `testreg.loadbalanced.includeServerPortInRequestURL` - {boolean true/false value which indicates if the port should be included to resolve the server}
- `testreg.loadbalanced.server.rest.contextpath` - {REST context name. e.g.: "/testreg.rest"}

- `testreg.loadbalanced.server.webapp.contextpath` - {webapp context name. e.g.: "/testreg.webapp". Leave this blank if you are using ROOT as webapp context name}

6.3.2.2 Database Configuration

Administration and Registration Tools (also known as ART) uses a MongoDB database. Follow the steps in Section 5 of this document to create a generic MongoDB server. ART will create a database with all required collections upon startup if one doesn't already exist.

6.3.2.3 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values.

```
export CATALINA_OPTS="-Xms512m -Xmx1024m -XX:PermSize=512m -XX:MaxPermSize=512m
-Dspring.profiles.active=progman.client.impl.integration,mna.client.integration
-Dprogman.baseUri=https://pm.example.org/rest/
-Dprogman.locator=Name,Environment -DmnaServerName=tib.example.org
-Djavax.net.ssl.trustStore=/trust/store/certs/path/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXXX"
```

6.3.2.4 Application Bootstrap

Uploading Accommodations and Designated Supports

As of Release R71, ART allows uploading of Accommodations and Designated Supports. This requires several steps:

1. Gather the required files into a directory

```
AccessibilityConfig.xml (from the repo)

AccessibilityConfig.xsd (from the repo)

validator.conf (from the repo)

validator.sh [for Linux/Mac] or validator.bat [for Windows] (from the repo)

AccommodationValidator.jar (needs to be built from source in AccValidation dir.)
```

2. Edit validator.conf to update the following information

```
basedir=.

validatorjar=AccommodationValidator.jar

accomconfig=AccessibilityConfig.xml

xsd=AccessibilityConfig.xsd

mongohost=host.foo.com

mongouser=muser

mongopwd=mpass
```

```
mongodbname=testreg
mongoport=27017
```

Update the parameters as required and save the file on a Linux box that has access to the ART database. This can also be done on a Windows box with similar access if necessary.

3. Execute the validator

***NOTE**: Prior to executing the validator, be sure to stop the Tomcat service that runs ART. This validator will update the ART DB so it should only be executed with ART not running.*

`validator.sh -c=validator.conf` [Linux/Mac] or

`validator.bat validator.conf` [Windows]

If this completes without issues, ART is ready to start.

ART Application Settings

In the running ART application there is a link at the top of the screen called Setting. There are two settings on that page that need to be set in order for the Data Warehouse integration to work correctly.

The first is `Test Reg System Id`. If nothing is set in the text field a button should be displayed next to the text field that will generate a random GUID to use as the system ID. Press the button to generate the GUID.

The second setting is `Assessment Eligibility types for export to Data Warehouse`. This is a multi-select list that allows for the setup of only specific types of assessments to control the Data Warehouse export. Select the types of assessments that should influence the export: summative, formative, and/or interim.

Be sure to save any settings changes that are made.

Field Name Crosswalk Setup

Each field that is part of an upload file has an associated field name that is displayed in various locations on the UI. It is possible to change the display names of the fields.

The initial mapping of the field names is kept in the REST module in the `/src/main/resources/spring/entity-crosswalk-context.xml` file. This can be modified and the WAR container restarted to pick up the changes.

Organizational Hierarchy Renaming (Entity)

The application allows for the renaming of the levels of the organizational (entity) hierarchy. For example, the lowest level of the hierarchy is an institution. Many deployments may wish to change this to always display "school" rather than "institution."

The renaming of the levels in the hierarchy is part of component branding in the program management component. Two steps are necessary: adding a tenant and then configuring the branding for the tenant and component.

If the deployment of ART is a state level deployment, all users who log into the system should ultimately have tenancy that resolves to the state for which the system has been deployed. If, however, the ART deployment truly is multi-tenanted, then this configuration must be performed for each tenant. Component branding is resolved from lower to higher in the tenant hierarchy.

Add a Tenant

In the Program Management application, choose Manage Tenants to create a tenant if one has not been created. Make sure that the tenant has a subscription to the ART component. If an ART component has not yet been created go to Manage Components to create one.

Configure Component Branding

To set up the names go to Configure Component Branding. All of the entries are of type `Property`. The list of properties to configure are:

- Assessment
- Assessments
- User
- Users
- Institution
- GroupOfDistricts
- GroupOfInstitutions
- GroupOfStates
- Student
- Students
- Institutions
- StudentGroups
- District
- State
- Accommodations
- Eligibility
- Student Group

For each property the value reflects what will be seen in the UI.

Other Miscellaneous Setup

In the ART UI there is a link at the top of the screen called Setting. The settings page has a few items that should be configured before users use the system:

- Client Name - The name of the top level Client for this deployment. This is bootstrapped from Program Management properties, but can be changed here.
- Time Zone - The effective time zone for this deployment. Choose from the drop down list.
- Share Student Identity Data - This checkbox turns on and off the choice to share student identity data. If selected, then student names, birth dates, and SSID will be shared out through external interfaces. If turned off, then this data will not be shared.
- Hide 'Group Of' Entity Levels - This must be configured before the system is used. Allows the system to completely hide the "Group of" levels of the organizational hierarchy. Once hidden, the system will not allow any functions to be performed for those levels of the hierarchy.

Other Dependencies and Progman Properties

There are many other properties that need to be set in Program Management so that ART will function correctly.

MNA Properties

- `testreg.mna.description` - Component name string to pass to M&A
- `mna.mnaUrl` - The URL of the M&A REST api (/rest/)
- `mna.oauth.batch.account` - Username (email address) of MNA client user used for authenticating into MNA and logging metrics information
- `mna.oauth.batch.password` - Password of MNA client user

RabbitMq Properties

- `rabbitmq.vhost` - The name of the RabbitMq ghost to connect to. The default vhost is "/", but RabbitMq could be configured to use something different.
- `rabbitmq.password` - Password to authenticate to RabbitMq with
- `rabbitmq.username` - Username to connect to RabbitMq as
- `rabbitmq.host` - RabbitMq server hostname

Security/SSO Properties

- `testreg.security.idp` - The URL of the SAML-based identity provider (OpenAM)
- `testreg.security.profile` - The name of the environment the application is running in. For a production deployment this will most likely be "prod". Must match the profile name used to name metadata files.

- `component.name` - The name of the component that this ART deployment represents. This must match the name of the component in Program Management and the name of the component in the Permissions application
- `permission.uri` - The base URL of the REST api for the Permissions application

Mongo Properties

- `testreg.mongo.hostname` - A comma delimited list of Mongo hostnames
- `testreg.mongo.port` - The Mongo port
- `testreg.mongo.username` - Username to connect to Mongo as
- `testreg.mongo.password` - Password for Mongo authentication
- `testreg.mongo.dbname` - The database name for the ART database

Other Miscellaneous Properties

- `client` - The name of the top-level client entity for this deployment
- `tsb.tsbUrl` - The base URL for the REST api of Test Spec Bank
- `language.codes` - Comma delimited list of valid language codes
- `testreg.minJs` - Flag for JavaScript minification, true to minify
- `testreg.rest.context.root` - The server relative context root for the ART REST WAR. Should start and end with a slash.
- `testreg.userguide.location` - The full URL to the ART User Guide.

6.3.3 Test Delivery System (TDS) Deployment (for both student and proctor)

Follow the steps outlined in Section 5 of this document (Server Creation) to create the TDS server.

6.3.3.1 Property Configuration - Proctor

Configuration settings for this component are defined via the Program Management application. Add all configuration settings based on the default set provided in the code repository [R10]. Please refer there for the latest information on setup and configuration.

Database Properties

The following parameters need to be configured inside program management for the database.

- datasource.url=jdbc:mysql://mysql.db.url:3306/schemaname - The JDBC URL of the database from which Connections can and should be acquired.
- datasource.username=<db-username> - Username that will be used for the DataSource's default getConnection() method.
- datasource.password=<db-password> - Password that will be used for the DataSource's default getConnection() method.
- datasource.driverClassName=com.mysql.jdbc.Driver - The fully qualified class name of the JDBC driverClass that is expected to provide Connections.
- datasource.minPoolSize=5 - Minimum number of Connections a pool will maintain at any given time.

- `datasource.acquireIncrement=5` - Determines how many connections at a time datasource will try to acquire when the pool is exhausted.
- `datasource.maxPoolSize=20` - Maximum number of Connections a pool will maintain at any given time.
- `datasource.checkoutTimeout=60000` - The number of milliseconds a client calling getConnection() will wait for a Connection to be checked-in or acquired when the pool is exhausted. Zero means wait indefinitely. Setting any positive value will cause the getConnection() call to timeout and break with an SQLException after the specified number of milliseconds.
- `datasource.maxConnectionAge=0` - Seconds, effectively a time to live. A Connection older than maxConnectionAge will be destroyed and purged from the pool. This differs from maxIdleTime in that it refers to absolute age. Even a Connection which has not had much idle time will be purged from the pool if it exceeds maxConnectionAge. Zero means no maximum absolute age is enforced.
- `datasource.acquireRetryAttempts=5` - Defines how many times datasource will try to acquire a new Connection from the database before giving up. If this value is less than or equal to zero, datasource will keep trying to fetch a Connection indefinitely.

MNA properties

The following parameters need to be configured inside program management for MNA.

- `mna.mnaUrl=http://<mna-context-url>/mna-rest/` - URL of the Monitoring and Alerting client server's rest url
- `mnaServerName=proctor_dev` - Used by the MNA clients to identify which server is sending the log/metrics/alerts.
- `mnaNodeName=dev` - Used by the mna clients to identify who is sending the log/metrics/alerts. There is a discrete mnaServerName and a node in case say XXX for server name & node1/node2 in a clustered environment giving the ability to search across clustered nodes by server name or specifically for a given node. It’s being stored in the db for metric/log/alert, but not displayed.
- `mna.logger.level=ERROR` - Used to control what is logged to the Monitoring and Alerting system. Logging Levels (ALL - Turn on all logging levels, TRACE, DEBUG, INFO, WARN, ERROR, OFF - Turn off logging).

SSO properties

The following parameters need to be configured inside program management for SSO.

- `permission.uri=https://<permission-app-context-url>/rest` - The base URL of the REST api for the Permissions application.
- `component.name=Proctor` - The name of the component that this Proctor deployment represents. This must match the name of the component in Program Management and the Permissions application.

- proctor.security.idp=https://<idp-url> - The URL of the SAML-based identity provider (OpenAM).
- proctor.webapp.saml.metadata.filename=proctor_sp.xml - OpenAM Metadata file name uploaded for environment and placed inside server directory.
- proctor.security.dir=file:////<sp-file-location-folder> - Location of the metadata file.
- proctor.security.saml.keystore.cert=<cert-name> - Name of the Keystore cert being used.
- proctor.security.saml.keystore.pass=<password> - Password for keystore cert.
- proctor.security.saml.alias=proctor_webapp - Alias for identifying web application.
- oauth.tsb.client=tsb - OAuth Client id configured in OAM to allow the SAML bearer workflow to convert a SAML assertion into an OAuth token for the "coordinated web service" call to TSB.
- oauth.access.url=https://<oauth-url> - OAuth URL to OAM to allow the SAML bearer workflow to POST to get an OAuth token for any "machine to machine" calls requiring OAUTH
- encrypt:oauth.tsb.client.secret=<password> - OAuth Client secret/password configured in OAM (under the client id) to allow the SAML bearer workflow to convert a SAML assertion into an OAuth token for the "coordinated web service" call to TSB.
- encrypt:mna.oauth.client.secret=<password> - OAuth Client secret/password configured in OAM to allow get an OAuth token for the "batch" web service call to MnA.
- mna.oauth.client.id=mna - OAuth Client id configured in OAM to allow get an OAuth token for the "batch" web service call to MnA.
- proctor.oauth.resource.client.secret=<password> - OAuth Client secret/password configured in OAM to allow get an OAuth token for the "batch" web service call to core standards.
- proctor.oauth.resource.client.id=proctor - OAuth Client id configured in OAM to allow get an OAuth token for the "batch" web service call to core standards.
- proctor.oauth.checktoken.endpoint=http://<oauth-url> - OAuth URL to OAM to allow the SAML bearer workflow to perform a GET to check that an OAuth token is valid.

Proctor properties

The following parameters need to be configured inside program management for Proctor

- proctor.IsCheckinSite=false
- proctor.ClientQueryString=false
- proctor.Appkey=Proctor
- proctor.EncryptedPassword=true
- proctor.RecordSystemClient=true
- proctor.SqlCommandTimeout=60
- proctor.AppName=Proctor
- proctor.SessionType=0 - Type of the testing supported: 0 is online, 1 is paper-based.
- proctor.DBJndiName=java:/comp/env/jdbc/sessiondb
- proctor.TestRegistrationApplicationUrl=http://url.to.art/ - URL to ART Application
- proctor.TDSArchiveDBName=archive - Name of the archive schema
- proctor.TDSSessionDBName=session - Name of the session schema
- proctor.TDSConfigsDBName=configs - Name of the config schema

- proctor.ItembankDBName=itembank - Name of the itembank schema
- proctor.Debug.AllowFTP=true
- proctor.StateCode=SBAC_PT
- proctor.ClientName=SBAC_PT

6.3.3.2 Property Configuration - Student

Configuration settings for this component are defined via the Program Management application. Add all configuration settings based on the default set provided in the code repository [R11]. Please refer there for the latest information on setup and configuration.

Database Properties

The following parameters need to be configured inside program management for the database:

- datasource.url=jdbc:mysql://localhost:3306/schemaname - The JDBC URL of the database from which Connections can and should be acquired.
- datasource.username=<db-username> - Username that will be used for the DataSource's default getConnection() method.
- encrypt:datasource.password=<db-password> - Password that will be used for the DataSource's default getConnection() method.
- datasource.driverClassName=com.mysql.jdbc.Driver - The fully-qualified class name of the JDBC driverClass that is expected to provide Connections
- datasource.minPoolSize=5 - Minimum number of Connections a pool will maintain at any given time.
- datasource.acquireIncrement=5 - Determines how many connections at a time datasource will try to acquire when the pool is exhausted.
- datasource.maxPoolSize=20 - Maximum number of Connections a pool will maintain at any given time.
- datasource.checkoutTimeout=60000 - The number of milliseconds a client calling getConnection() will wait for a Connection to be checked-in or acquired when the pool is exhausted. Zero means wait indefinitely. Setting any positive value will cause the getConnection() call to timeout and break with an SQLException after the specified number of milliseconds.
- datasource.maxConnectionAge=0 - Seconds, effectively a time to live. A Connection older than maxConnectionAge will be destroyed and purged from the pool. This differs from maxIdleTime in that it refers to absolute age. Even a Connection which has not been much idle will be purged from the pool if it exceeds maxConnectionAge. Zero means no maximum absolute age is enforced.
- datasource.acquireRetryAttempts=5 - Defines how many times datasource will try to acquire a new Connection from the database before giving up. If this value is less than or equal to zero, datasource will keep trying to fetch a Connection indefinitely.

MNA properties

The following parameters need to be configured inside program management for MNA.

- mna.mnaUrl=http://<mna-context-url>/mna-rest/ - URL of the Monitoring and Alerting client server's rest URL.

- mnaServerName=student_dev - Used by the mna clients to identify which server is sending the log/metrics/alerts.
- mnaNodeName=dev - Used by the mna clients to identify who is sending the log/metrics/alerts. There is a discrete mnaServerName and a node in case say XXX for server name & node1/node2 in a clustered environment giving the ability to search across clustered nodes by server name or specifically for a given node. It's being stored in the db for metric/log/alert, but not displayed.
- mna.logger.level=ERROR - Used to control what is logged to the Monitoring and Alerting system. The logging levels are ALL - Turn on all logging levels, TRACE, DEBUG, INFO, WARN, ERROR, OFF - Turn off logging.

Student properties

The following parameters need to be configured inside Program Management for Student.

- student.IsCheckinSite=false
- student.DONOT_Distributed=true
- student.ClientName=SBAC_PT
- student.StateCode=SBAC_PT
- student.ClientQueryString=true
- student.ClientCookie=true - If it is turned on, Student will try to get clientname from cookie
- student.Appkey=Student
- student.EncryptedPassword=true
- student.RecordSystemClient=true
- student.AdminPassword=<password> - Admin Password
- student.AppName=Student - Application name
- student.SessionType=0 - Type of the testing supported: 0 is online, 1 is paper-based
- student.DBDialect=MYSQL - Indicates which database we are using
- student.TestRegistrationApplicationUrl=http://localhost:8083/ - URL to TR(ART) Application
- student.TDSArchiveDBName=archive - Name of the archive schema
- student.TDSSessionDBName=session - Name of the session schema
- student.TDSConfigsDBName=configs - Name of the config schema
- student.ItembankDBName=itembank - Name of the itembank schema
- student.Debug.AllowFTP=true
- student.TDSReportsRootDirectory=/usr/local/tomcat/resources/tds/tdsreports/ - Directory on Student server box where TDS reports generated after student finished the test are located.
- student.StudentMaxOpps=2

6.3.3.3 Database Configuration

The TDS student and proctor applications use the same database. For production systems it is recommended to use large clustered instances such as Amazon's RDS. On the database machine, the following settings must be made.

_key	name	description	homepath
25	SBAC	(NULL)	/tmp/1/
26	SBAC_PT	(NULL)	/path/to/resources/tds/

Create Default Databases

The following steps will create the DBs and all necessary objects within. There are also drop scripts: drop_constraints.sql and drop_tables.sql should there be need to drop all the tables and start over. All scripts mentioned below are located at tdsdll project, tds-dll-schemas module.

'configs' Database

- Run the following command on the db server: `CREATE DATABASE 'configs' DEFAULT CHARACTER SET=utf8`
- Create tables by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/configs/create_tables.sql
- Create constraints by running the SQL script file create_tables.sql located at tds-dll-schemas/src/main/resources/sql/MYSQL/configs/create_constraints.sql
- Create indexes by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/configs/create_indexes.sql
- Create stored procedures by running the SQL script file/files located in folder tds-dll-schemas/src/main/resources/sql/MYSQL/configs/StoredProcedures/
- Create functions by running the SQL script file/files located in folder tds-dll-schemas/src/main/resources/sql/MYSQL/configs/Functions/

'itembank' Database

- Run the following command on the db server: `CREATE DATABASE 'itembank' DEFAULT CHARACTER SET=utf8`
- Create tables by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/itembank/create_tables.sql
- Create constraints by running the SQL script file create_tables.sql located at tds-dll-schemas/src/main/resources/sql/MYSQL/itembank/create_constraints.sql
- Create indexes by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/itembank/create_indexes.sql
- Create triggers by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/itembank/Triggers/triggers.sql
- Create stored procedures by running the SQL script file/files located in folder tds-dll-schemas/src/main/resources/sql/MYSQL/itembank/StoredProcedures/
- Create functions by running the SQL script file/files located in folder tds-dll-schemas/src/main/resources/sql/MYSQL/itembank/Functions/

'session' Database

- Run the following command on the db server: `CREATE DATABASE 'session' DEFAULT CHARACTER SET=utf8`
- Create tables by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/session/create_tables.sql

- Create constraints by running the SQL script file create_tables.sql located at tds-dll-schemas/src/main/resources/sql/MYSQL/session/create_constraints.sql
- Create indexes by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/session/create_indexes.sql
- Create triggers by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/session/Triggers/triggers.sql
- Create views by running the SQL script file/files located in folder tds-dll-schemas/src/main/resources/sql/MYSQL/session/Views/
- Create stored procedures by running the SQL script file/files located in folder tds-dll-schemas/src/main/resources/sql/MYSQL/session/StoredProcedures/
- Create functions by running the SQL script file/files located in folder tds-dll-schemas/src/main/resources/sql/MYSQL/session/Functions/

'archive' Database

- Run the following command on the db server: `CREATE DATABASE 'archive' DEFAULT CHARACTER SET=utf8`
- Create tables by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/archive/create_tables.sql
- Create indexes by running the SQL script file located at tds-dll-schemas/src/main/resources/sql/MYSQL/archive/create_indexes.sql

Load Configuration Data

1. Run the below three files on `configs` db to setup generic configs db:
   `/tds-dll-schemas/src/main/resources/import/genericsbacconfig/gen1.sql`
   `/tds-dll-schemas/src/main/resources/import/genericsbacconfig/gen2.sql`
   `/tds-dll-schemas/src/main/resources/import/genericsbacconfig/gen3.sql`
2. Run these statements on `itembank` db to setup generic itembank db:

```
INSERT INTO `itembank`.`tblclient`
(`name`,
`description`,
`homepath`)
VALUES
('SBAC_PT',
NULL,
/*IMP: place root directory for the items path here */);
b. INSERT INTO `itembank`.`tblitembank`
(`_fk_client`,
`homepath`,
`itempath`,
`stimulipath`,
`name`,
`_efk_itembank`,
`_key`,
`contract`)
```

```
VALUES
(1, /*should be equal to _key column from tblclient tbl for this client name */
/*IMP: place relative home path for this itembank here */,
/*IMP: place the item path here */,
/*IMP: place the stimuli path here */
NULL,
1,
1,
NULL);
```

Example:

```
Insert into `itembank`.`tblclient` (`name`, `description`,`homepath`)
values ('SBAC', null, '/usr/local/tomcat/resources/tds/');
insert into INTO
`itembank`.`tblitembank`(`_fk_client`,`homepath`,`itempath`,`stimulipath`,`name`,`_efk_itemba
nk`,`_key`,`contract`)
values (1, 'bank/', 'items/', 'stimuli/', null, 1, 1, null);
```

Load Test Package into the Database

Execute/Run stored procedure `loader_main()` in database 'itembank' once per testpackage file. The stored proc takes one input, that is the XML file content. Copy the XML file content and paste it as an input and execute the stored procedure. Sample test packages are available in the **assessmentpackages** project. For example:

```
Call `itembank`.`loader_main` ('<testpackage purpose="administration" publisher="SBAC_PT"
publishdate="Aug 15 2014  9:19AM" version="1.0">.. .. .</testpackage>')
```

6.3.3.4 Server startup configuration settings

Create or edit a file named `setenv.sh` under the `$CATALINA_HOME/bin` directory and define startup environmental variables. Here is a sample setenv.sh file; replace all key values with appropriate values. These Tomcat/JVM values have been shown to be effective in a high performance scenario; modify if needed:

```
# TDS JVM config For Higher Performance
JAVA_OPTS="$JAVA_OPTS -server"
JAVA_OPTS="$JAVA_OPTS -Xms1080m"
JAVA_OPTS="$JAVA_OPTS -Xmx804m"
JAVA_OPTS="$JAVA_OPTS -XX:NewSize=484m"
JAVA_OPTS="$JAVA_OPTS -XX:MaxNewSize=512m"
JAVA_OPTS="$JAVA_OPTS -XX:PermSize=512m"
JAVA_OPTS="$JAVA_OPTS -XX:MaxPermSize=512m"
JAVA_OPTS="$JAVA_OPTS -Dsun.io.useCanonCaches=false"
JAVA_OPTS="$JAVA_OPTS -Dsun.net.client.defaultConnectTimeout=40000"
JAVA_OPTS="$JAVA_OPTS -Dsun.net.client.defaultReadTimeout=60000"

# Garbage Collection
JAVA_OPTS="$JAVA_OPTS -verbose:gc"
```

```
JAVA_OPTS="$JAVA_OPTS -Xloggc:/usr/local/tomcat/logs/gc.log"
JAVA_OPTS="$JAVA_OPTS -XX:+PrintClassHistogram"
JAVA_OPTS="$JAVA_OPTS -XX:+PrintGCTimeStamps"
JAVA_OPTS="$JAVA_OPTS -XX:+PrintGCDetails"
JAVA_OPTS="$JAVA_OPTS -XX:+UseParNewGC"
JAVA_OPTS="$JAVA_OPTS -XX:+UseConcMarkSweepGC"
JAVA_OPTS="$JAVA_OPTS -XX:+UseCMSCompactAtFullCollection"
JAVA_OPTS="$JAVA_OPTS -XX:+CMSClassUnloadingEnabled"
JAVA_OPTS="$JAVA_OPTS -XX:+DisableExplicitGC"

# Heap Dumps
JAVA_OPTS="$JAVA_OPTS -XX:+HeapDumpOnOutOfMemoryError"
JAVA_OPTS="$JAVA_OPTS
-XX:HeapDumpPath=/usr/local/apache-tomcat-7.0.41/logs/heapdump.hdprof"
```

Note:
Xms and Xmx sizing should be at least one third of the server's RAM size but not more than 3/4 of the server's RAM size. Assigning a heap/RAM size below 500M to Xms/Xms may not work very well with this web application. We recommend having a minimum of 1GB RAM on a server for this application.
XX:NewSize, NewRatio, MaxNewSize, PermSize, MaxPermSize should range between a minimum of 20% to a maximum of 50% of the total size of -Xms. Other Parameters can remain as shown.

TDS-specific configurations:

```
export CATALINA_OPTS="-DSB11_CONFIG_DIR=$CATALINA_HOME/resources
-Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JLogger
-Dlog4j.configuration=file:///path/to/tomcat/conf/log4j.xml
-Dspring.profiles.active=progman.client.impl.integration,mna.client.integratio
n -Dprogman.baseUri=http://url.to.progman.rest.interface/
-Dprogman.locator='Proctor,envname;Student,envname'
-DmnaServerName=name-of-this-tds-server
-Djavax.net.ssl.trustStore=/path/to/tomcat/ssh/.truststore.aws
-Djavax.net.ssl.trustStorePassword=XXXXXXXXXXXXXX"
```

- Run the Jenkins deploy script to push a version of student (or proctor) to this machine.
- Add the `vvt` mime type to Tomcat's web.xml file for the student site. This enables support for closed-captioning.

Please refer to the *Test Delivery repository* [R10] for more information.

6.3.3.5 IRiS

IRiS (Item Rendering System) is a component within the Student repository which provides a web service to render content using the same control paths that the student application uses except that it is very lightweight. This rendered content can be embedded in an iFrame for purposes such as item review. The IRiS jar can be deployed in its own Tomcat-based Linux web server as described in Section 5.3.

Deploying Content to IRiS

The easiest way to deploy content is to scp the file to the server. Content needs to be deployed to the folder `/usr/local/tomcat/content`

Loading Content

Once content has been deployed as above, the system should pick it up automatically. However, this feature may not be reliable - especially when deploying massive number of files by copying from a remote location. The only reliable way to make sure the system picks up all newly deployed content is to hit this API endpoint:

```
/iris/Pages/API/content/reload
```

This is a blocking call and is an initial attempt at providing an API that doesn't involve restarting the server.

IRiS Read-only Mode

The sample page endpoint `/iris/IrisPages/sample.xhtml` and scripts that have been supplied put the site in read-only mode such that user actions on the interface will not change the response. The response, however, may be programmatically set to something else as is done when "Set Response" button is clicked.

6.3.4 Adaptive Engine Deployment

This component is an implementation of an adaptive algorithm. Similar to the AI Scoring Engine component, this component will have performance issues if implemented as a network service instead of a Test Delivery component plugin. The Test Spec Bank contains metadata that defines the algorithm or engine to use. The Test Delivery Component is expected to load the correct algorithm or engine when the test is being administered.
This component is automatically deployed when deploying the TDS system (Section 6.3.3.).

6.3.5 Item Scoring Engine Deployment

Item Scoring [R18] is comprised of a set of modules that programmatically score items in real time while the student is taking the test. It uses standard QTI response processing with custom operators, an extension point. This engine supports response processing on the following item scoring categories: Trivially scored, Machine scored, and Machine scored with custom operators. Refer to the custom operators document for more information on this.

The item scoring service is normally deployed on the same server as TDS, but this is not a requirement. Each item type can be configured to point to either a local or remote scoring module. Item Scoring also supports asynchronous scoring, and may also be called by the TIS system (see 6.4) for post-assessment scoring of "unscored" items. Configuration information for the various modules is available at [R18].

6.3.6 Dictionary Deployment

The Dictionary subsystem integrates with the Student application to provide access to tools/accessibility features such as dictionary, glossary, and thesaurus. It consists of the UI and implementation logic required for calling the Merriam-Webster REST API and transforming the response.

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Tomcat server. The dictionary service may be deployed on an existing Tomcat container (e.g. the TDS server), or on a server of its own, depending on performance requirements.

6.3.7 Dictionary Provisioning

Configuration information for the various modules is available at [R19]. Please refer there for the latest information on setup and configuration.

6.3.7.1 Property Configuration

The following parameters need to be set inside of the Tomcat server's context.xml on the Dictionary server:

```
<Parameter name="tds.dictionary.key.TDS_Dict_Collegiate" override="false" value="---Key ---"/>
<Parameter name="tds.dictionary.url.TDS_Dict_Collegiate" override="false"
value="http://www.dictionaryapi.com/api/v1/references/collegiate/xml/"/>
<Parameter name="tds.dictionary.key.TDS_Dict_Learners" override="false" value="---Key ---"/>
<Parameter name="tds.dictionary.url.TDS_Dict_Learners" override="false"
value="http://www.dictionaryapi.com/api/v1/references/learners/xml/"/>
<Parameter name="tds.dictionary.key.TDS_Dict_SD2" override="false" value="---Key ---"/>
<Parameter name="tds.dictionary.url.TDS_Dict_SD2" override="false"
value="http://www.dictionaryapi.com/api/v1/references/sd2/xml/"/>
<Parameter name="tds.dictionary.key.TDS_Dict_SD3" override="false" value="---Key ---"/>
<Parameter name="tds.dictionary.url.TDS_Dict_SD3" override="false"
value="http://www.dictionaryapi.com/api/v1/references/sd3/xml/"/>
<Parameter name="tds.dictionary.key.TDS_Dict_SD4" override="false" value="---Key ---"/>
<Parameter name="tds.dictionary.url.TDS_Dict_SD4" override="false"
value="http://www.dictionaryapi.com/api/v1/references/sd4/xml/"/>
<Parameter name="tds.dictionary.key.thesaurus" override="false" value="---Key ---"/>
<Parameter name="tds.dictionary.url.thesaurus" override="false"
value="http://www.dictionaryapi.com/api/v1/references/ithesaurus/xml/"/>
<Parameter name="tds.dictionary.key.spanish" override="false" value="---Key ---"/>
<Parameter name="tds.dictionary.url.spanish" override="false"
value="http://www.dictionaryapi.com/api/v1/references/spanish/xml/"/>
```

Note that the **seven** dictionary keys must be obtained and licensed from Merriam-Webster. Smarter Balanced vendors may contact Smarter Balanced for more information.

The following parameter is optional in context.xml:

```
<Parameter name="tds.dictionary.groupName" override="false" value="SBAC"/>
```

If `groupName` for the Dictionary project is specified, then an appropriate database entry in the TDS database must be made. To do this, execute the following script in TDS:

```
INSERT INTO `configs`.`tds_applicationsettings` (`_key`, `environment`, `appname`, `property`, `type`, `isoperational`) VALUES ('660B3EAF-993B-4289-A2C9-28AB72B7B2CH', 'Development', 'Student', 'tds.dictionary.group', 'string', 1);


INSERT INTO `configs`.`system_applicationsettings` (`clientname`, `environment`, `appname`, `property`, `type`, `value`, `servername`, `siteid`, `_fk_tds_applicationsettings`)
VALUES ('SBAC', 'Development', 'Student', 'tds.dictionary.group', 'string', 'SBAC', '', '',(select _key from configs.tds_applicationsettings where property like '%tds.dictionary.group%'));
```

Additionally, this parameter must be added to the Program Management configuration for the Student component:

`tds.testshell.dictionaryUrl=http://<host>/Dictionary` - URL for the Dictionary project deployment.

6.3.8 Computer Adaptive Testing (CAT) Simulator Deployment

The web simulator allows authorized users to configure and simulate tests for a specified number of times on specified number of opportunities. This software provides a web interface to configure and manage simulations and run simulations asynchronously by spawning worker threads. Configuration information is available at [R20]. Please refer there for the latest information on setup and configuration.

6.3.9 Computer Adaptive Testing (CAT) Simulator Provisioning

- Set up the necessary itembank, config and session MySQL databases for simulation
- Load test packages necessary for simulation into the itembank
- User need to have an account to run simulations. Accounts are created by authorized users in the session database used for simulation.
- Change the settings.xml in tds.websim.web.ui module to specify the itembank, config and session databases to be used with the simulator and the database credentials.

6.4 Assessment Scoring

Deployed Component → / Dependent component ↓	Test Integration System (TIS)	Teacher Hand Scoring System (THSS)
Single Sign On (SSO)	Y	Y
Admin and Registration Tool	Y	
TDS / IRiS		Y
TDS	Y	
Test Integration System (TIS)	n/a	Y
Teacher Hand Scoring System (THSS)	Y	n/a
Data Warehouse	Y	
Database server	MS-SQL	MS-SQL

Table 4: Assessment Scoring Component Dependencies

6.4.1 Machine / AI Scoring Deployment

Machine scoring is an integral part of the Test Delivery system and is automatically deployed when deploying the TDS system (Section 6.3.3.). Refer also to Section 6.3.5 for Item Scoring information.

6.4.2 Student Report Processor Deployment

The Student Report Processor (SRP) is a JAR which generates student XML reports and submit to TIS (Test Integration System). It should be set up to run as a background process. It processes records in its queue one by one. The XML reports associated with the test opportunities for those records are sent to TIS and then deleted. Configuration information is available at [R21]. Please refer there for the latest information on setup and configuration.

In general, build the code and deploy the JAR file. Run the jar file as background process. Create a startup script to run the application on machine reboot and another script to monitor the application and start it up if it is not running.

```
/usr/bin/java \
  -DreportingDLL.tisUrl="<TIS URL>" \
  -DreportingDLL.tisStatusCallbackUrl="<TIS Status Callback URL>" \
```

```
  -Doauth.tis.client.id="<Oauth Client ID>" \
  -Doauth.tis.client.secret="<Oauth Client Secret>" \
  -Doauth.tis.username="<Oauth Username>" \
  -Doauth.tis.password="<Oauth Password>" \
  -Doauth.access.url="<Oauth Access URL>" \
  -Dlogfilename="/path/to/log/file/maintenance" \
  -Djdbc.driver="com.mysql.jdbc.Driver" \
  -Djdbc.url="jdbc:mysql://name.of.mysql.server:3306/session" \
  -Djdbc.userName="<mysql_username>" \
  -Djdbc.password="<mysql_password>" \
  -DDBDialect="MYSQL" \
  -classpath ".:/jar/file/path/to/tdsMaintenance.jar:/mysql/connector/path/mysql-connector-java-5.1.22-bin.jar"
org.opentestsystem.delivery.studentreportprocessor.StudentReportProcessor
```

6.4.2.1 Database Configuration

For populating academicYear inside the XML Report, configure schoolyear column according to current year inside `itembank.tbltestadmin` table. This column needs to be updated every year.

Example:

```
UPDATE `itembank`.`tbltestadmin` SET `schoolyear`='2015' WHERE `_key`='SBAC';
UPDATE `itembank`.`tbltestadmin` SET `schoolyear`='2015' WHERE `_key`='SBAC_PT';
```

6.4.3 Test Integration System (TIS) Deployment

Since hand-scored items go through a different process than machine-scored items, this component takes the machine-scored items of a test and integrates them with the hand- scored items of the same test. After scoring is complete, the data is then uploaded into the Data Warehouse.

6.4.4 Test Integration System (TIS) Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Windows/IIS server. A typical production deployment might consist of 2 web servers and 2 DB servers. Each server's specs would be similar to the following: 4 vCPU, 15GB RAM, 40GB system, 150GB data. Single partition on each disk, .NET version 4.5.2+, IIS version 7.5+. Please refer to the *Test Integration System repository* [R16] for more information.

6.4.4.1 Database Configuration

TIS uses MS-SQL 2008 (not MySQL) as its server back end. The recommended server specifications are AWS EC2 m3.2xlarge.

To begin, create the following databases on the [DB Server]:

- `OSS_TIS`
- `OSS_Itembank`
- `OSS_Configs`
- `OSS_TestScoringConfigs`

- ```
  - TIS Service will need an account with R/W/X to OSS_TIS and R/X to the other 3 dbs
  - TIS services REST endpoint will need an account with R/W/X access to OSS_TIS
  - The Scoring Daemon will need an account with R/W/X access to OSS_TIS
  ```

Deploy the `[Db server].OSS_TIS` database objects by running the following scripts in order: [DB server]

- `<root>\TDSQAService\OSS.TIS\SQL\TISDB\1_Create_Objects.sql`
- `<root>\TDSQAService\OSS.TIS\SQL\TISDB\2_Configuration.sql`
- `<root>\TDSQAService\OSS.TIS\SQL\TISDB\3_ScoringDaemonConfiguration.sql` (** this script will require a couple of variables to be set prior to running)

Deploy the `[Db server].OSS_TestScoringConfigs` database objects by running the following scripts in order:

- `<root>\TDSQAService\OSS.TIS\SQL\TestScoringConfigs\1_Tables.sql`
- `<root>\TDSQAService\OSS.TIS\SQL\TestScoringConfigs\2_Views.sql`
- `<root>\TDSQAService\OSS.TIS\SQL\TestScoringConfigs\3_Configuration.sql`

Deploy the `[Db server].OSS_Configs` database objects by running the following scripts in order:

- `<root>\TDSQAService\OSS.TIS\SQL\TDSConfigs\1_Create_Objects.sql`
- `<root>\TDSQAService\OSS.TIS\SQL\TDSConfigs\2_Configuration.sql`

Deploy the `[Db server].OSS_Itembank` database objects by running the following scripts in order:

- `<root>\TDSQAService\OSS.TIS\SQL\TDSItemBank\1_Create_Synonyms_Sproc.sql`
- `<root>\TDSQAService\OSS.TIS\SQL\TDSItemBank\2_Create_Synonyms_Config.sql` (** this script will require a couple of variables to be set prior to running)
- `<root>\TDSQAService\OSS.TIS\SQL\TDSItemBank\3_Create_Objects.sql`
- `<root>\TDSQAService\OSS.TIS\SQL\TDSItemBank\4_Configuration.sql`
- `<root>\TDSQAService\OSS.TIS\SQL\TDSItemBank\5_LoadPackages.sql`

6.4.4.2 Server Startup Configuration settings

Create these folders on the application server (if they don't already exist): [App server]

- `/Services/tis_opentestsystem`
- `/oss_tis_itemscoring`
- `/oss_tisservices`

Deploy TISService code at `tis_opentestsystem/` [App server]

To deploy the 'TIS Scoring Daemon' -

- Create a web application on the App server at `/oss_tis_itemscoring'
- Publish the 'TIS Scoring Daemon' application to `/oss_tis_itemscoring`

To deploy the 'TIS services REST endpoint' -

- Create a web application on the App server at `/oss_tisservices'
  ```

- Publish the 'REST endpoint application' to `/oss_tisservices`

Run InstallUtil for .Net 4.5, 32-bit on `/tis_opentestsystem/TDSQAService.exe` to install the windows service. [App server]

Verify that the TIS service has privileges to write to the event log [App server].

6.4.5 Teacher Hand Scoring System (THSS) Deployment

This component provides the interface humans use to score items and view rubrics on how to score the items even when the scorers are not centrally located. It also delivers those scores back to the Test Delivery and Data Warehouse components to be stored with the student responses. It also allows for the development of machine scoring capabilities that are used in conjunction with the human scoring capabilities. It should be able to use the AI Scoring engine(s) that are used in adaptive testing by Test Delivery.

6.4.6 Teacher Hand Scoring System (THSS) Provisioning

Please refer to the *Configuration/Provisioning Guide* section in this document to set up a new generic Windows/IIS server. Please refer to the *Teacher Hand Scoring System repository* [R17] for more information.

6.5 Post-Deployment Configuration

6.5.1 Security

- X509 certificates will need to be installed on each load balancer to secure communications. References:
    - Updating a certificate on the load balancer.
    - Using HTTP/HTTPS Protocol with Elastic Load Balancing.

7 Bootstrapping the System

The system consists of components which may be deployed at multiple tenancy levels (e.g., Consortium / top level, state level, etc.). As an integrated system, however, it is important to follow the following sequence when bootstrapping the system.

1. OpenAM or the selected SSO solution
2. Program Management
3. Monitoring and Alerting
4. Permissions
5. Test Item Bank and Core Standards
6. Portal and Digital Library

The remainder of the components may be deployed in any order once the shared services are available.

7.1 Creating the Prime User

While users are generally provisioned in ART, it's not possible to do so without logging into ART with a user - hence a chicken-and-egg problem. In order to provision a "Prime User" for a new system, an XML file must be generated and copied into the SSO system's dropbox for consumption. A sample XML is available in the *Administrative* repository [R1] under the `pm` directory. An example is below:

```
<?xml version='1.0' encoding='UTF-8'?>
<Users>
<User Action="ADD">
  <UUID>CREATE-UNIQUE_UUID_HERE</UUID>
  <FirstName>CHOOSE-FIRST_NAME</FirstName>
  <LastName>CHOOSE-LAST_NAME</LastName>
  <Email>temp-bootstrap@example.com</Email>
  <Phone/>
  <Role>
    <RoleID></RoleID>
    <Name>Administrator</Name>
    <Level>CLIENT</Level>
    <ClientID>CHOOSE-CLIENT_IDENTIFIER_NUMBER</ClientID>
    <Client>CHOOSE-UNIQUE_CLIENT_NAME</Client>
    <GroupOfStatesID/>
    <GroupOfStates/>
    <StateID/>
    <State/>
    <GroupOfDistrictsID/>
    <GroupOfDistricts/>
    <DistrictID/>
    <District/>
```

```
    <GroupOfInstitutionsID/>
    <GroupOfInstitutions/>
    <InstitutionID/>
    <Institution/>
  </Role>
</User>
</Users>
```

WARNING: The Prime User is a *temporary* administrative user that is pushed directly into SSO. Since the user is not created with ART, it cannot be managed within ART. The idea is to use the Prime User for all necessary logins until ART is up and running, then create a new user via ART with Administrator role. At that point, *it is recommended that this Prime User be deleted for security reason*s.

7.1.1 Field Explanations and Notes

- **UUID**: This must be a unique identifier within the SSO system. Generate a UUID/GUID using any available mechanism (no dashes, lowercase OK).
- **FirstName** and **LastName**: Select this user's first and last names. These fields are essentially irrelevant, but it's a good idea to use a descriptive name for the first and last names.
- **Email**: The email address does not have to be real, but does have to be unique. This serves as the user's username.
- **Role/Name**: This field must contain "Administrator".
- **Role/Level**: This field must contain "CLIENT".
- **Role/ClientId**: This should be a unique text string identifier for the client, usually numeric.
- **Role/Client**: This should be a unique text string identifier for the client name. Selection of this name is important down the road when performing ART configuration. *This name cannot be changed once the Prime User is inserted into SSO* and configuration of other applications has begun.

7.1.2 Inserting Prime User Into SSO

Once the XML file is created, name it `prime_user_testfile_.xml`. The `_testfile_` extension ensures that an email will not be sent to the user's email (since it is fake). Place this file into the SSO FTP dropbox site. You will need the URL, username, and password of the SFTP site in order to do this. The prime user's default password is *password1*. The value of the default password is set in the User Provisioning script on the OpenDJ server (`/opt/scripts/sbacProcessXML.pl`). The extension used to signify "do not send emails" is configured in Program Management's configuration file for ART.

7.1.3 Protected Roles

The Administrator role is usually considered a special ("protected") role ("with great power comes great responsibility"). It's possible to set and unset the "protected" status of any role via Permissions. It's important to note that only users with protected roles can even view those protected roles in ART; therefore when initially setting up a system, it's more convenient to not

set the protected status of any of the roles until the system has been properly configured and regular users are ready to be provisioned.

7.2 Creating the Entity Hierarchy

Users and Students must be associated to an entity in order to exist in the system. Table 7.1 below describes the entity hierarchy levels in the SmarterApp system. The “Group Of…” entities, shown in dotted lines, are optional.

While the prime user is associated with the top-level (Client) entity, users and students are usually associated with lower-level entities (Institutions and Districts, for example). The entity hierarchy levels are fixed as shown (other than the optional elements), but users and students can be added and managed using the ART application (see 6.3). When new users are added to the system, the entity to which they are associated must already exist. For example, if student Johnny Appleseed is to be added to School “Generic High”, an institution with that name must already exist, and be parented in some way to the Client entity. Usually this is done by creating the parent District, its parent State, and parenting that state to the Client.

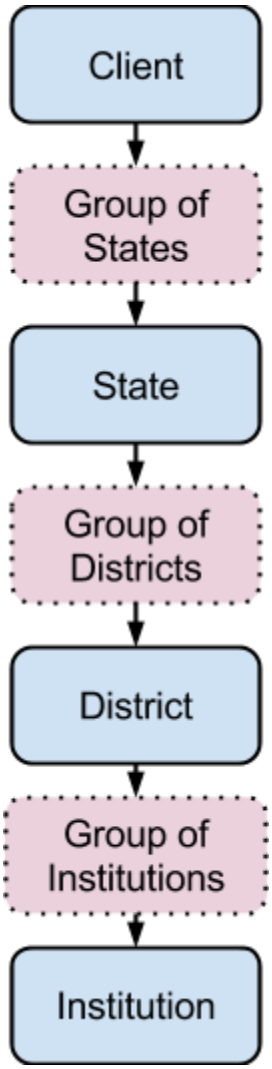


Table 7.1: SmarterApp Entity Level Hierarchy

7.2.1 The Tenancy Mechanism

Client components, such as TDS or ART, use a tenancy mechanism for authorization which depends on three main shared services: SSO, Program Management, and Permissions. This mechanism is used to manage User Attributes, Roles and Tenancy Chains, Roles to component Permissions mappings, and Tenants and Component Subscriptions.

Prior to a user logging on to a component such as Proctor, the component needs to know:

- **What roles matter to me, and what are their mappings to my permissions?**
    - This information is obtained via a REST call to Permissions. It might find out that users with *Test Administrator* roles have permissions to proctor summative tests, while users with *Interim Test Administrator* roles have permission to proctor only non-summative tests.
- **What tenants have a valid subscription to me?**
    - This information is obtained via a REST call to Program Management. It might find out that MN, ID, and CA have valid subscriptions to Proctor, but KS does not.

Next, after the user logs on, SSO returns the following information about the user:

- **Is this user authenticated?**
    - Is the username/password combination valid?
- **Does this user have any roles that matter to me?**
    - Role information is returned in the SAML response
- **Does this user belong to a valid tenant?**
    - Tenancy information is returned in the SAML response
- **Does the tenant have a current subscription to me?**
    - Tenant subscription information was obtained in the previous step from Program Management
- **What permissions does this user have?**
    - The user's roles are mapped against the permissions obtained earlier.
- **What entity at what level is this user's role(s) associated with?**
    - Role and Tenancy information from the user is mapped against the information obtained earlier.
- **What is the lineage of the entity?**
    - See Table 7.1.

Based on these answers, the component:

- Resolves role conflicts if necessary
- Determines associated entity and tenant
- Determines what features the user can access
- Determines what data to show the user
- Optionally brand itself with component assets

7.3 Creating Additional Users

Once the Administrator user has been created, it will be possible to log into any application as long as the proper permissions have been configured (see the Permissions section, under Database configuration). To ensure a newly created user has access to a particular application, several steps must be taken in these running apps.

7.3.1 Program Management

The Program Management component is used to:

1. Create *component*(s), for example the Applications in your system.
2. Create *tenant*(s) of which the user will be a member, if not already there (for example, State or District entities).
3. Ensure the *tenants* are subscribed to the *component*(s), are in good standing, and are within the appropriate effective date range (for example, the state of AK is subscribed to the Program Management, Permissions, and ART components).
4. Create a *configuration file* for the new component, and ensure the component references it properly (including environment name). Refer to the Property Configuration section of each component.

7.3.2 Permissions

1. Create a *component* with the exact same name as the one created in ProgMan, if not already there.
2. Create a new *role*, if not already there
3. Add a *permission* for that component, if not already there
4. Map the *role* to the component *permission*(s).

7.3.3 ART

1. Create a new *user* in ART application (must be configured for provisioning new users into OAM).
2. Assign one or more roles (defined in Permissions) to that user.
3. Associate the user with an entity that is in the hierarchy of the tenant which you defined in ProgMan.

A detailed matrix of component dependencies is available at Reference [3].