Production environment

Overview

The purpose of this guide to highlight key points that should be considered during installation of the platform in production environments.

Since target environment may vary greatly depending on your preferences of Linux the exact installation instructions will vary. If you would like some advice or help with installation to specific environment stacks please contact us using feedback form

Tomcat

Connector configuration

In order to work with large import files such as bulk image import upload file limits have to be adjusted.

This can be configured in **server.xml** (usually located in $CATALINA_HOME/conf/) by adjusting **maxPostSize** parameter on the connectors.

Example of changing upload size to 100MB

```
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000" maxPostSize="104857600"
               redirectPort="8443"
               URIEncoding="UTF-8"/>

    <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               connectionTimeout="20000" maxPostSize="104857600"
               keystoreFile="conf/ssl/sslkey" keystorePass="ycselfsigned"
               clientAuth="false" sslProtocol="TLS" URIEncoding="UTF-8"/>
```

> It is specific to your business what the maximum size should be. You need to investigate this and decide upon a reasonable size. We

recommend to set this at no less than 100MB.

Please note that connector must have **UTF-8** encoding enabled.

Context.xml

Platform apps use context.xml configuration for database connection. These are kept in **META-INF/context.xml** files in each deployable war file (i.e. yes-manager.war, yes-api.war and ROOT.war for Admin, REST and SF respectively).

The files are preprocessed during maven build and are populated with correct value for target environment (see Maven profiles and variable section in From source guide).

We highly recommend using Tomcat 7.0.35+ as earlier versions have know issues with loading META-INF/context.xml incorrectly.

If this is not possible some workarounds are:

- Copy the contents of the files into main context file **$CATALINA_BASE/conf/context.xml**
- Copy the file **yes-manager.war/META-INF/context.xml** to **$CATALINA_BASE/conf/Catalina/localhost/yes-manager.xml**, **yes-shop.war/META-INF/context.xml** to **$CATALINA_BASE/conf/Catalina/localhost/yes-shop.xml** and **yes-api.war/META-INF/context.xml** to **$CATALINA_BASE/conf/Catalina/localhost/yes-api.xml**

For more information on contexts please consult Tomcat Documentation

Configuring AJP connector

There are two supported options to setup tomcat:

- Tomcat + AJP + Apache HTTP
- Tomcat only

> The major difference is in how the SSL certificates are setup and how redirection between http and https pages is done.

Without going into deep discussion we recommend "***Tomcat + AJP + Apache***". This means that the SSL communication is done at the Apache HTTP level leaving the Tomcat server just to perform the processing duties. The main reason for this is that AJP cannot be configured with SSL support it is only able to redirect to the secure port when the https connection is required.

Admin is preconfigured to use only HTTPS and YeS secure page configuration is fully managed by the Wicket framework.

We do not provide any specific guides since this is beyond the scope of this document. There are plenty of OS specific guides on the Internet. However in a nutshell the configuration steps are:

- Configure Tomcat + AJP + Apache HTTP. (see Tomcat documentation https://tomcat.apache.org/tomcat-7.0-doc/config/ajp.html)
- Setup Apache HTTP SSL support. (see Apache documentation http://httpd.apache.org/docs/2.2/ssl/ssl_faq.html)
- Block all Tomcat ports to prevent it being accessed directly
- Ensure that YeS war is build with ssl profile switched off (default configuration)

> If you are using Tomcat on its own you you need to use **ssl** profile during maven build and configure SSL certificate for Tomcat

Wicket configuration (Tomcat without Apache HTTP)

ssl profile influences the following configurations:

Sets transport mode in web.xml

WEB-INF/web.xml security configurations

```
...
   <security-constraint>
       <!-- ADM should only be accessible via https -->
       <display-name>ADM Security</display-name>
       <web-resource-collection>
           <web-resource-name>ADM</web-resource-name>
           <url-pattern>/*</url-pattern>
       </web-resource-collection>
       <user-data-constraint>
           <transport-guarantee>CONFIDENTIAL</transport-guarantee>
       </user-data-constraint>
   </security-constraint>
...
```

Configures redirect ports in Wicket application

org.yes.cart.web.application.StorefrontApplication class

```
 ...
        if ("true".equalsIgnoreCase(getInitParameter("secureMode"))) {

            final HttpsConfig httpsConfig = new HttpsConfig(
                    Integer.valueOf((String)
ObjectUtils.defaultIfNull(getInitParameter("unsecurePort"), "8080")),
                    Integer.valueOf((String)
ObjectUtils.defaultIfNull(getInitParameter("securePort"), "8443"))
            );

            final HttpsMapper httpsMapper = new
HttpsMapper(getRootRequestMapper(), httpsConfig);

            setRootRequestMapper(httpsMapper);

        }
...
```

Above configurations can be customised via environment specific property files:

```
# Configurations used when -Pssl is used
YC_HOME/env/maven/TARGETENVIRONMENT/config-tomcat-ssl-on.properties
# Configurations used when ssl profile is not used
YC_HOME/env/maven/TARGETENVIRONMENT/config-tomcat-ssl-off.properties
```

Here are two interesting articles from the Wicket wiki that provide more insight:

https://cwiki.apache.org/confluence/display/WICKET/Wicket+application+behind+mod_proxy_http+and+https

https://cwiki.apache.org/confluence/display/WICKET/How+to+switch+to+SSL+mode

Apache HTTP

VirtualHosts for shop instances URLs

> Configuring Virtual hosts from scratch is beyond the scope of this document. Please refer to documentation for detailed steps: http://httpd.apache.org/docs/2.2/vhosts/examples.html

An example configuration for a virtual host with single Tomcat instance hosting storefront and Admin server looks like this:

Example configuration

```
...
<VirtualHost *:80>
        ServerAdmin webmaster@yes-cart.org
 ServerName yes-cart.org
 ServerAlias *.yes-cart.org
        ProxyIOBufferSize 65536
 ProxyRequests Off
 <Proxy *>
  Order deny,allow
  Allow from all
 </Proxy>

 ProxyPass /yes-shop/ ajp://localhost:8009/
        ProxyPassReverse /yes-shop/ ajp://localhost:8009/

 ProxyPass /yes-manager/ ajp://localhost:8009/yes-manager/
        ProxyPassReverse /yes-manager/ ajp://localhost:8009/yes-manager/

        DocumentRoot /var/html/yes-cart.org

        <Directory />
  Options FollowSymLinks
  AllowOverride None
 </Directory>
 <Directory /var/html/yes-cart.org/>
  Options Indexes FollowSymLinks MultiViews
  AllowOverride All
  Order allow,deny
  allow from all
 </Directory>
 Redirect permanent / http://www.yes-cart.org/yes-shop/
</VirtualHost>
...
```

However for production configuration we recommend setting up at least three Tomcat instances:

- Admin server Tomcat instance
- Storefront primary Tomcat instance
- Storefront secondary Tomcat instance

Storefront primary and secondary should be configured via load balancer (which can be done in Apache HTTP configuration)

> See more detailed information on http://httpd.apache.org/docs/2.2/mod/mod_proxy_ajp.html

Special configurations

Admin to storefront (SF) communication

Admin application needs to be aware of the SF nodes in order to trigger various functions such as full text re-indexing, cache evictions, health monitoring etc.

All cluster communication is configured in

```
YC_HOME/env/maven/TARGETENVIRONMENT/config-cluster.properties
```

Specifically **cluster.config.protocol** can be set to use the protocol that you require.

Default configuration is **WS** to use web service communication is is unidirectional from Admin to SF.

IPv4 and **IPv6** configurations use multicasting (backed by JGroups), which require additional setup and configuration of **yc-jgroups-udp.xml** file. Multicasting allows bi-directional communication and less network overhead but requires more configurations.

Regardless of the communication type the purpose of the channel is to allow Admin application send messages to SF and REST webapps.

Web Service cluster communication strategy

Web Service uses URLs that each node specifies in **META-INF/context.xml** and **cluster.xml** files for node discovery both of which are setup by maven when resources are processed. URLs can be configured using environment specific property file:

```
YC_HOME/env/maven/TARGETENVIRONMENT/config-cluster.properties
```

> By default cluster.xml uses 3 nodes ADM, YES0 and YES1 (REST API) if additional nodes are needed this file has to be updated. We recommend following current setup and add placeholders which will be filled by maven using config-cluster.properties

When each web app context is started these configurations are loaded by node service. ADM node uses "ping()" call to determine which nodes specified in cluster.xml are online. Note that full clear cache request will force ADM to "re-ping()" all nodes. So if some nodes came online late clearing all cache should help ADM re-discover the cluster.

> See "System > Cluster" section in ADM to view these settings

JGroups multicasting cluster communication strategy

It is possible to configure multicasting communication strategy by specifying IPv4 or IPv6 strategy depending of what IP address format your network uses.

All configurations are located in:

```
YC_HOME/env/maven/TARGETENVIRONMENT/config-cluster.properties
```

JGroups specific configurations are located in yc-jgroups-udp.xml file which is preprocessed by maven to fill in node specific variables. yc-jgroups-udp.xml follows standard JGroups multicasting configuration.

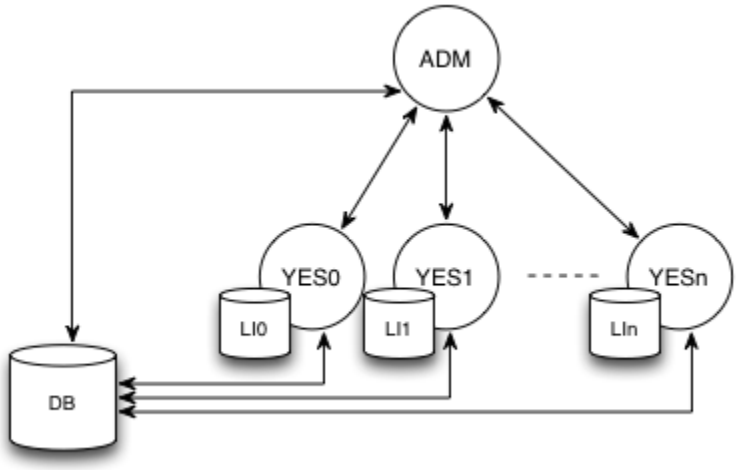
> Because JGroups is additional dependency when IPv4 or IPv6 strategy is used the build must be done with **jgroups** Maven profile enabled

Clustering

All applications are java webapps clustering approach follows standard practice under servlet containers, such as Tomcat.

The recommended approach configuration is to keep a single Admin instance and several SF/Rest instances in the following manner:

Basic Cluster



Database clustering is transparent to all web apps as they access via same URI, which could in itself point to single DB instance or DB cluster.

Admin app needs to be aware of all SF apps and thus cluster communication channel has to be configured (as mentioned in previous section).

Each node (webapp) in cluster stores its discovery information in context.xml file which is then detected by the communication channel so that Admin web app is able to broadcast notifications to SF nodes.

Admin context.xml configuration

```
<Parameter name="NODE_ID" value="JAM" override="false"/>
 <Parameter name="NODE_TYPE" value="ADM" override="false"/>
 <Parameter name="NODE_CONFIG" value="DEFAULT" override="false"/>
 <Parameter name="CLUSTER_ID" value="YCCLUSTER" override="false"/>
```

SF node 0 context.xml configuration

```
  <Parameter name="NODE_ID" value="YES0" override="false"/>
   <Parameter name="NODE_TYPE" value="SFW" override="false"/>
   <Parameter name="NODE_CONFIG" value="DEFAULT" override="false"/>
   <Parameter name="CLUSTER_ID" value="YCCLUSTER" override="false"/>
   <Parameter name="CHANNEL_URI"
value="http://localhost:8080/yes-shop/services/backdoor" override="false"/>
```

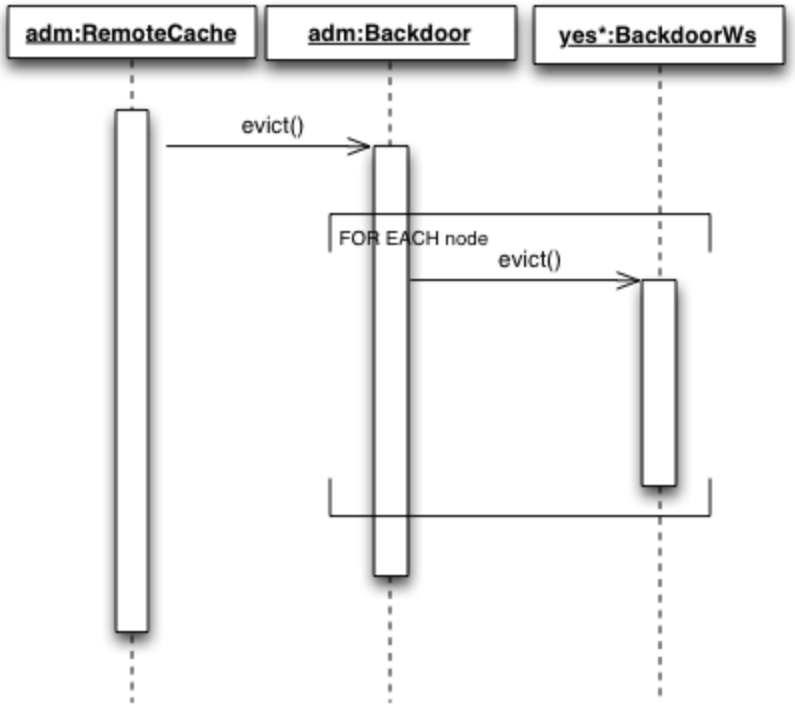
SF node 1 context.xml configuration

```
  <Parameter name="NODE_ID" value="YES1" override="false"/>
   <Parameter name="NODE_TYPE" value="API" override="false"/>
   <Parameter name="NODE_CONFIG" value="DEFAULT" override="false"/>
   <Parameter name="CLUSTER_ID" value="YCCLUSTER" override="false"/>
   <Parameter name="CHANNEL_URI"
value="http://localhost:8081/yes-shop/services/backdoor" override="false"/>
```

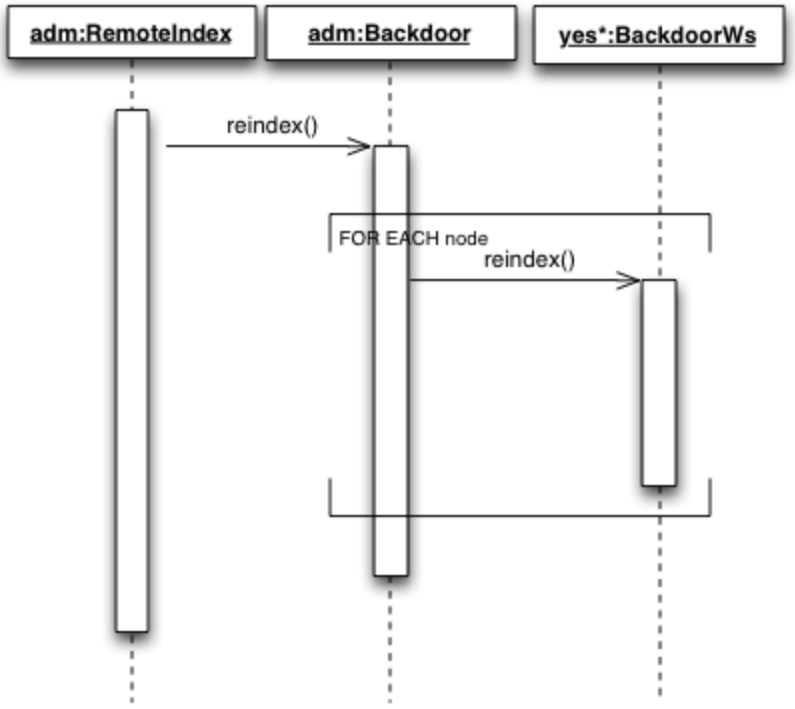
SF node n context.xml configuration

```
  <Parameter name="NODE_ID" value="YESn" override="false"/>
   <Parameter name="NODE_TYPE" value="SFW" override="false"/>
   <Parameter name="NODE_CONFIG" value="DEFAULT" override="false"/>
   <Parameter name="CLUSTER_ID" value="YCCLUSTER" override="false"/>
   <Parameter name="CHANNEL_URI"
value="http://localhost:808n/yes-shop/services/backdoor" override="false"/>
```

Cache eviction through WS

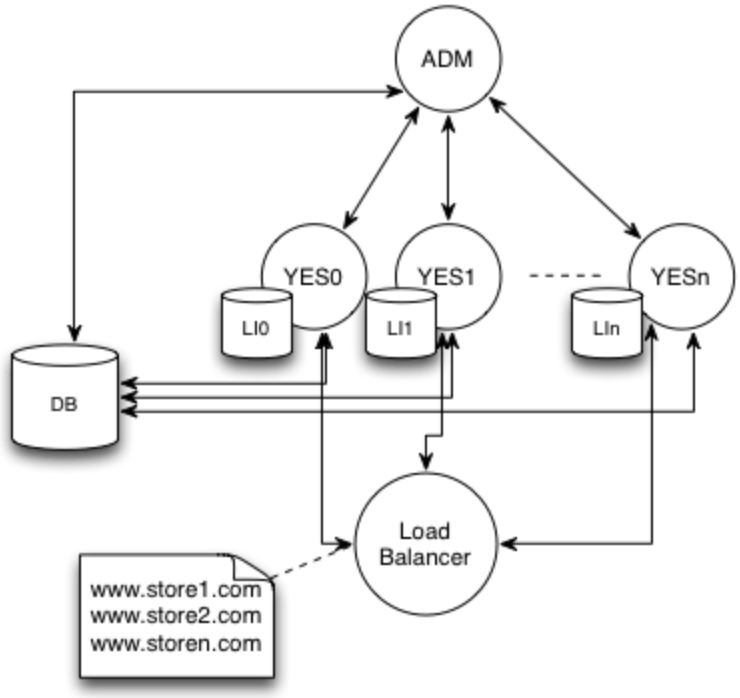


Indexing through WS

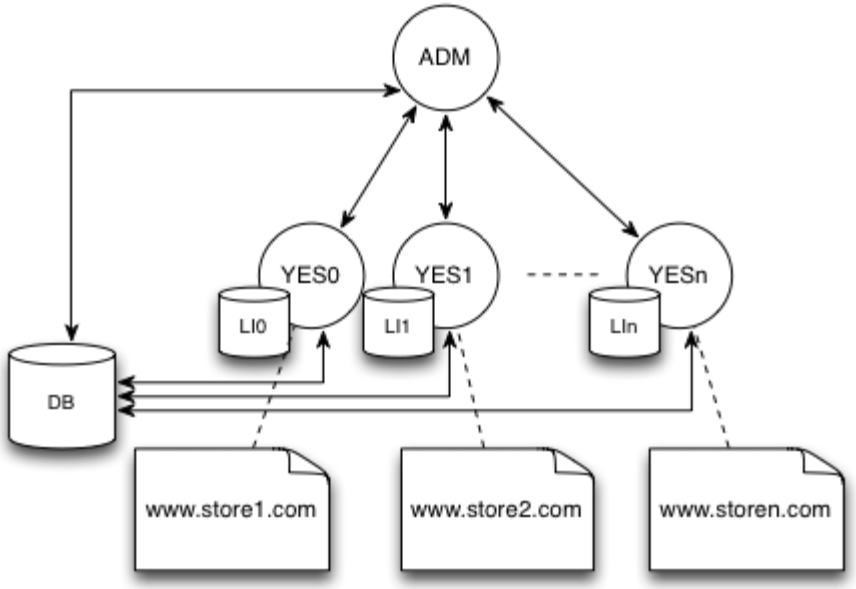


There are several strategies to SF nodes configuration

Load balancer - each node can serve request for any store instance on the cluster



Node per store - each node serves requests for specific store instance (only for multi store environments)



Topology of the cluster is highly dependent on the business requirements.

We strongly advise to keep things simple and only create complex cluster when it is necessary.

Full text indexing

Out of the box Full text search is facilitated by embedded Lucene, which means that a dedicated FT index is created for each web app as it needs exclusive file system IO access to write to indices. SaaS has option to use FT connector web app that allows to share FT index between several applications.

Inventory data relevancy

There are some caveat to clustered SF architecture with dedicated FT indices as they may become out of sync due to parallel inventory updates on different nodes. In pursue of high throughput the platform design has gone on path of high performance as opposed to high inventory accuracy when browsing the website and viewing product details pages (checkout still uses fresh from DB data). We anticipate that this adverse effect will only be valid for use cases for low stock high demand products or discontinued items.

To overcome this deficiency the following scheduled jobs are employed

Job	Purpose	Default schedule	Config location
Reindex Discontinued Products	Remove discontinued products from FT index	every day 5am	web/support/src/main/resources/websupport-cronjob.xml
Reindex All Products	Full reindex	disabled	web/support/src/main/resources/websupport-cronjob.xml
Inventory Changes Product Indexing	Reindex products for which inventory changed since last execution	every 5 minutes	web/support/src/main/resources/websupport-cronjob.xml

Generally **Inventory Changes Product Indexing** job should be enough to keep data fresh. However **Reindex All Products** can be used as a replacement for **Reindex Discontinued Products** to recreate index from time to time.

More advanced solutions involve using low stock thresholds (through custom attributes) to prevent out of stock items.

This issue is however only related to displaying of data since checkout process uses fresh from database data to check inventory levels. Therefore you can never oversell. The customer will simply be presented with an out of stock message.

Cache considerations

Platform apps use Ehcache to facilitate caching. All caches operate with two parameters settings:

- timeToLiveSeconds - maximum time the cache is allowed to exists regardless of its use
- timeToIdleSeconds - time to when cache becomes idle and hence expired.

E.g. if we have timeToLiveSeconds = 3600 (1h) and timeToIdleSeconds = 60 (1m) and no one accesses the object for more than one minute the cache will expire. If however the cache gets used with frequency less than a minute it will expire after 1h.

These parameters will need to be adjusted according to your website needs. The longer cache times used the more throughput the website will have but at the cost of slightly outdated data being displayed, thus the right balance of cache timeouts needs to be established that your customers are comfortable with.

The platform does not use Hibernate second level cache as it in fact worsens the performance. This is due to the fact that before caching the platform does a lot of preprocessing and thus keeps only ready to use objects in cache as opposed to raw persistence data. If second level cache is enabled Hibernate transforms all queries into two phase retrieval process: 1) select ids to get list of primary keys 2) select each item in the list by id, which causes substantial increase in number of queries and hence contingency on database. Therefore it is highly recommended not to use second level cache.

MySQL tips

Hibernate connection exception due to time out

MySQL has a timeout on connection and sometimes when the server is left idle (e.g. overnight) the first user to enter the site sees a DB connection exception. Then everything works fine. THIS IS NOT PLATFORM ISSUE! This problem is specific to MySQL timeouts that Hibernate cannot detect. To fix it you need a ping query in context.xml so that MySQL does not drop the connection without hibernate knowing about it.

In order to enable ping query additional parameters are added to context.xml's Resources **validationQuery="SELECT 1" autoReconnect="true"**, which are filled in by db.config.yes.custom and db.config.yespay.custom variable from config-db-mysql.properties

Path to context.xml's:

```
# YC-SHOP
YC_HOME/web/store-wicket/src/main/webapp/META-INF/context.xml
# REST API
YC_HOME/web/api/src/main/webapp/META-INF/context.xml
# ADM
YC_HOME/manager/server/src/main/webapp/META-INF/context.xml
```

Example YC SHOP context.xml configuration with specified attributes

```
<Resource name="jdbc/yespayjndi"
                  auth="Container"
                  scope="Shareable"
                  type="javax.sql.DataSource"
                  maxActive="1000"
                  maxIdle="30"
                  maxWait="10000"
                  removeAbandoned="true"
                  validationQuery="SELECT 1"
                  autoReconnect="true"
                  username="app"
                  password="app"
                  driverClassName="org.apache.derby.jdbc.ClientDriver"
                  url="jdbc:derby://localhost:1527/yespay"
                  minEvictableIdleTimeMillis="864000000" />

<Resource name="jdbc/yesjndi"
                  auth="Container"
                  scope="Shareable"
                  type="javax.sql.DataSource"
                  maxActive="100"
                  maxIdle="30"
                  maxWait="10000"
                  removeAbandoned="true"
                  validationQuery="SELECT 1"
                  autoReconnect="true"
                  username="app"
                  password="app"
                  driverClassName="org.apache.derby.jdbc.ClientDriver"
                  url="jdbc:derby://localhost:1527/yes"
                  minEvictableIdleTimeMillis="864000000" />
```

Cloud computing

Cloud computing is very large topic for discussion. Mainly it depends on the cloud offering. Some clouds are very unpredictable in the way they behave such as cutting threads if resources are low, inability to access file system etc. Therefore there is no single solution for all clouds and the

integrations are very much vendor specific. Therefore tuning platform to specific Cloud is very much manual process.

We do provide AWS provisioning script and a step by step guide as an example.