

|  |
| --- |
| Solution Architecture Document |
| <ClientID>-<ProductID> |

| REVISION HISTORY | | | | | |
| --- | --- | --- | --- | --- | --- |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| n.n |  |  | dd-mmm-yyyy |  | dd-mmm-yyyy |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

| Related Artifacts | |
| --- | --- |
| Ref. | Name |
|  |  |
|  |  |

Contents

[1 Executive Summary 5](#_Toc461542037)

[2 Introduction 6](#_Toc461542038)

[Definitions, Acronyms, Abbreviations 6](#_Toc461542039)

[Purpose 6](#_Toc461542040)

[2.1.1 Stakeholders 6](#_Toc461542041)

[2.1.2 GOALS 6](#_Toc461542042)

[2.1.3 SCOPE 7](#_Toc461542043)

[2.1.4 MAJOR BUSINESS DRIVERS 7](#_Toc461542044)

[3 Requirements 9](#_Toc461542045)

[Stakeholders 9](#_Toc461542046)

[Functional requirements 10](#_Toc461542047)

[3.1.1 Roles in the system 10](#_Toc461542048)

[3.1.2 Web UI wireframe 10](#_Toc461542049)

[Non-functional requirements 14](#_Toc461542050)

[3.1.3 Self service: UI 14](#_Toc461542051)

[Constraints 14](#_Toc461542052)

[Assumptions 14](#_Toc461542053)

[4 Quality Attributes 16](#_Toc461542057)

[Considered Solution Architecture Quality Attributes 16](#_Toc461542058)

[Architecturally Significant Quality Attributes 16](#_Toc461542059)

[5 Baseline Solution architecture 17](#_Toc461542060)

[High-level solution structure 17](#_Toc461542061)

[5.1.1 DEPLOYMENT SERVER 17](#_Toc461542062)

[5.1.2 INITIAL SETUP PACKAGE 18](#_Toc461542063)

[5.1.3 COLLABORATION SPACE (NOT IN SCOPE OF MVP 19](#_Toc461542064)

[5.1.4 SELF-SERVICE NODE 19](#_Toc461542065)

[5.1.5 EXPLORATORY ENVIRONMENT 21](#_Toc461542066)

[SOLUTION IN AWS CLOUD 23](#_Toc461542067)

[5.1.6 Azure mapping 23](#_Toc461542068)

[Solution components structure 24](#_Toc461542069)

[5.1.7 SELF-SERVICE UI COMPONENT 24](#_Toc461542070)

[5.1.8 PROVISION SERVICE 25](#_Toc461542071)

[5.1.9 BACKEND 25](#_Toc461542072)

[5.1.10 Authentication and Authorization approach 25](#_Toc461542073)

[5.1.11 DB 26](#_Toc461542074)

[Domain model 27](#_Toc461542075)

[Data model 27](#_Toc461542076)

[5.1.12 Storage 1 27](#_Toc461542077)

[5.1.13 Storage 2 27](#_Toc461542078)

[5.1.14 … 27](#_Toc461542079)

[High-level deployment approach 27](#_Toc461542080)

[Architecturally Significant Quality Attributes 27](#_Toc461542081)

[5.1.15 Security 27](#_Toc461542082)

[5.1.16 Supportability 28](#_Toc461542083)

[5.1.16.1 Monitoring 28](#_Toc461542084)

[5.1.16.2 Administration Tools 28](#_Toc461542085)

[5.1.16.3 Specific deployment aspects 28](#_Toc461542086)

[6 Target Solution Architecture 29](#_Toc461542087)

[Risks 29](#_Toc461542088)

[dependencies 29](#_Toc461542089)

[Assumptions 29](#_Toc461542090)

[High-level solution structure 29](#_Toc461542091)

[6.1.1 Solution Component 1 29](#_Toc461542092)

[6.1.2 Solution Component 2 30](#_Toc461542093)

[6.1.3 … 30](#_Toc461542094)

[Solution components structure 30](#_Toc461542095)

[6.1.4 Solution Component 1 30](#_Toc461542096)

[6.1.5 Solution Component 2 30](#_Toc461542097)

[6.1.6 … 31](#_Toc461542098)

[Domain model 31](#_Toc461542099)

[Data model 31](#_Toc461542100)

[6.1.7 Storage 1 31](#_Toc461542101)

[6.1.8 Storage 2 31](#_Toc461542102)

[6.1.9 … 31](#_Toc461542103)

[7 Technology stack 31](#_Toc461542104)

[High-level deployment approach 32](#_Toc461542105)

[Architecturally Significant Quality Attributes 32](#_Toc461542106)

[7.1.1 Security 32](#_Toc461542107)

[7.1.2 Supportability 33](#_Toc461542108)

[7.1.2.1 Monitoring 33](#_Toc461542109)

[7.1.2.2 Administration Tools 33](#_Toc461542110)

[7.1.2.3 Specific deployment aspects 33](#_Toc461542111)

[Architecture Constraints and Limitations 33](#_Toc461542112)

[8 Transition 33](#_Toc461542113)

[Appendix A. Solution Architecture Quality Attributes 35](#_Toc461542114)

# Executive Summary

[**Description**: This is a mostly non-technical summary of the entire Solution Architecture Document (SAD) for customer top management. We recommend creating it based on the sections below. The idea was to prepare a ready-to-use template for this document type.

Section Type: Mandatory]

# Introduction

Current project is infrastructure deployment agnostic platform for data analytics, science and machine learning. Platform greatly simplifies and reduces costs for infrastructure preparation and configuration, thus provides with possibility to kick-off project in short-term.

Platform can be deployed on various cloud and hypervisor providers. Currently in scope of MVP AWS platform support shall be implemented, meanwhile Azure support is planned for next release.

Future roadmap is to fully support major public and private cloud providers.

## Definitions, Acronyms, Abbreviations

[**Description:** This section must clearly clarify all the definitions, acronyms and abbreviations used in the document.

**Section Type:** Highly recommended]

|  |  |
| --- | --- |
| Abbreviation or Acronym | Definition |
|  |  |
|  |  |

## Purpose

[**Description:** This section explains why this architectural work is being done. This provides the basis for the “executive summary” section.

You need list stakeholders of the architectural work and goals of this work.

Section Type: Mandatory]

### Stakeholders

[**Description:** You need list stakeholders of the architectural work, their roles, names, contact data and their reasons for requesting this architectural work in the table below.

Section Type: Mandatory]

| Stakeholder Side | Stakeholder Role | Stakeholder Name | Contacts | Architectural Work Request Reasons |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

### GOALS

Major goals are to build:

**Self-service infrastructure**

* Users in charge of provisioning their own environment.
* No dedicated IT Support time required

**Fail-safe environment**

* sandbox - you can break it
* built within client’s security perimeter
* client’s security policies and access controls applied

**Exploratory Environment**

* best open source data tools
* scalable compute
* data and metadata store
* training / workshops

**Collaborative Data Science Workflow**

* collaboration within data science team
* collaboration between data science and engineering team
* fast track from RND to Production

**Customizable**

* Plug-in proprietary components

### SCOPE

In scope of MVP following features are planned:

* Full support of AWS cloud provider
* Automated creation/management of AWS infrastructure
* Automated provision/management of Exploratory Environment:
  + Jupyter with both Scala and Python kernel support
  + Local Apache Spark computation tool based on EMR libraries
  + Provision of EMR cluster with Spark
  + Provision of private storage based on S3 buckets
* Implement web-based user interface for Exploratory Environment Management.

### MAJOR BUSINESS DRIVERS

**Problem with DATA ACCESS**

* No access to production data
* Restricted access to production and raw data
* Data access requires IT assistence

**Problem with TOOLS**

* Only sql-based tools avaliable on the cluster (RDBMS, hive, impala, etc…)
* Latest tools avaliable only on local machines – difficult cooperation within a team
* Different tools used by data science and engineering teams – difficult cooperation between teams
* Instalation of tools difficult and time consuming (compex security policies, no experience)
* Tools present, but unstable (misconfiguration, insuficient permission, etc...)

**Problem with SUPPORT**

* 1 full time operation engineer required for each 2-3 data scientist (for efficient workflow)
* Data science team restricted to local machines, due to lack of IT support time
* Data science team relies on continuous support of IT ops team in most of daily activities

# Requirements

[**Description:** This section lists key high-level requirements (use-cases or scenarios from the use-case model), if:

* They represent some significant, central functionality of the final system.
* They have a large architectural coverage (they exercise many architectural elements).
* Or they stress or illustrate a specific, delicate point of the architecture.

Address all assumptions in sections 3.1-3.3.

Note: It is good practice to have requirements in a separate document/system in case of regular project activity. In this case, the author of the SAD document must reference the requirements in this document instead of duplicating them. It still makes sense to keep short requirement descriptions in the Solution Architecture Document in case of time-limited pre-sale activity.

**Section Type:** Mandatory (if chapter sections are not covered with available solution documentation)]

## Stakeholders

[**Description:** This section lists solution architecture stakeholders. This activity must be done before processing requirements. Stakeholders are individuals or groups who have and impact on or will be impacted by the solution architecture.

Each stakeholder of a solution — customer, user, project manager, coder, tester, and so on — is concerned with different characteristics of the system that are affected by its architecture. For example:

The user is concerned that the system is fast, reliable, and available when needed.

The customer is concerned that the architecture can be implemented on schedule and according to budget.

The manager is worried (in addition to concerns about cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways.

The architect is worried about strategies to achieve all of those goals.

Concerns are crucially important to the stakeholders of the solution, and determine its acceptability.

You need to identify all stakeholders correctly, communicate with them in order to understand their concerns and translate the concerns in requirements.

Consider the following stakeholders:

* Customer side:
  + Government Regulator
  + Sponsor
  + Product Owner
  + Project Manager
  + Operational Support Engineer
  + End User
  + Business Analyst
  + Solution Architect
  + Developer
  + Quality Assurance Engineer
* EPAM Side:
  + Account Manager
  + Project Manager
  + Business Analyst
  + Solution Architect
  + Developer
  + Quality Assurance Engineer
  + Operational Support Engineer

**Section Type:** Mandatory (if section is not covered with available documentation)]

| Stakeholder Side | Stakeholder Role | Stakeholder Name | Key concern | Requirements area |
| --- | --- | --- | --- | --- |
|  | Sponsor, Product Owner | Valentin Tsitlik |  |  |
|  | Product owner, End user | Paul Gesiak |  |  |

## Functional requirements

[**Description:** This section describes key requirements/use-cases (from the customer’s point of view and the Solution Architect’s point of view).

**Section Type:** Mandatory (if section is not covered with available documentation)]

### Roles in the system

Following system roles are currently envisioned:

* Administrator / DevOps
  + Performs initial infrastructure setup.
  + Have SSH access to Self-Service node, thus can change Self-Service configuration.
* Data Scientist
  + Can create / manage own exploratory environment via Self-Service Web UI.
  + Can run perform data exploration via exploratory environment Jupyter UI.

### Web UI wireframe

Web UI is expected to be used by desktop users on different platform and with different browsers. That’s why an expectation is to get responsible web UI which would provide equal experience for Windows, Mac and Linux users in the following browsers:

* MS IE 10+
* Google Chrome 42+
* Firefox 35+
* Safari 9.1.2+ only on Mac

The login page is very simple and allows use to be validated against LDAP, look at Figure 1

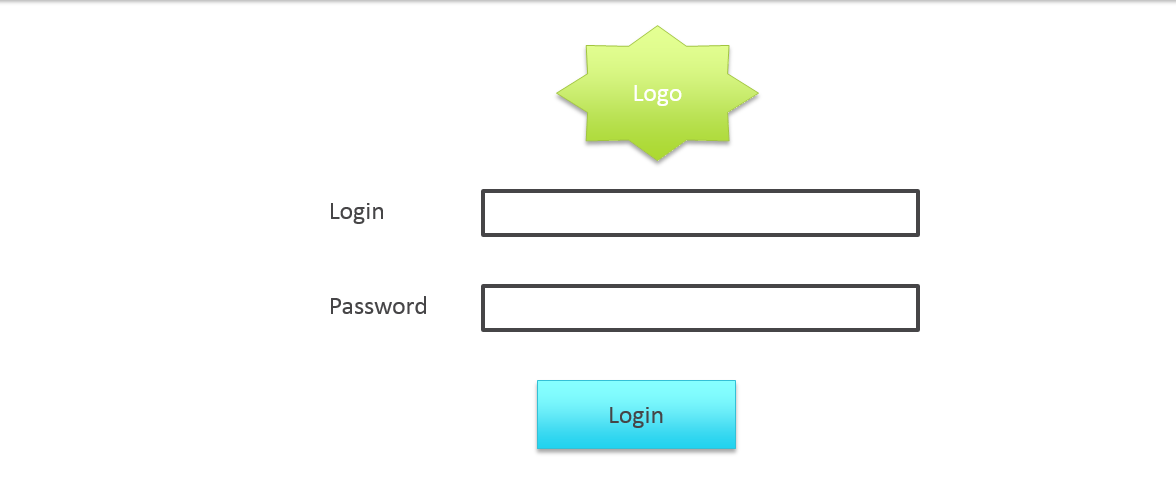


Figure 1

When user logs in first time (Figure 2), he/she expected to be asked to provide public SSH key and only after the environment will be created (in this save, bucket on S3, subnet in VPN).

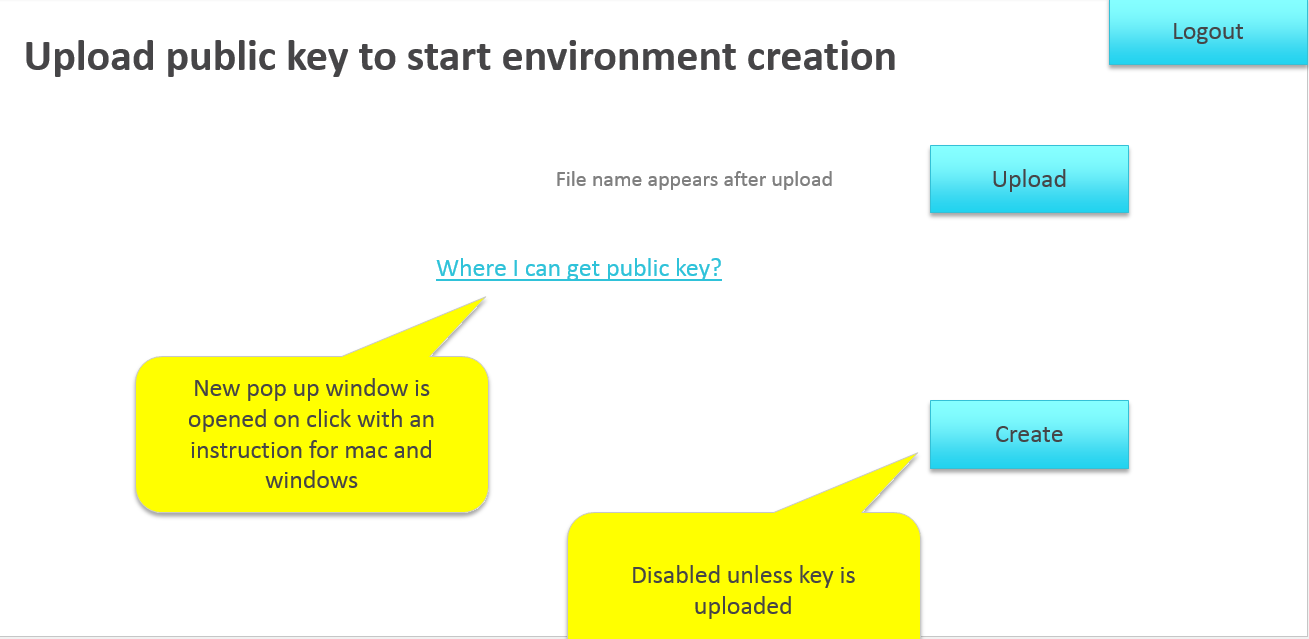


Figure 2

A button “Create” is inactive until key is loaded.

There is a link “Where I can get public key?” which opens pop-up with instruction how to generate public/private SSH keys on Windows and Mac and what exactly expected from user. It must be cleat for not very technical guy. After clicking “Create” button it must be some “wait until load” message which will wait backend to perform all preparation steps and redirect user to page Figure 3 List resources when it’s ready.

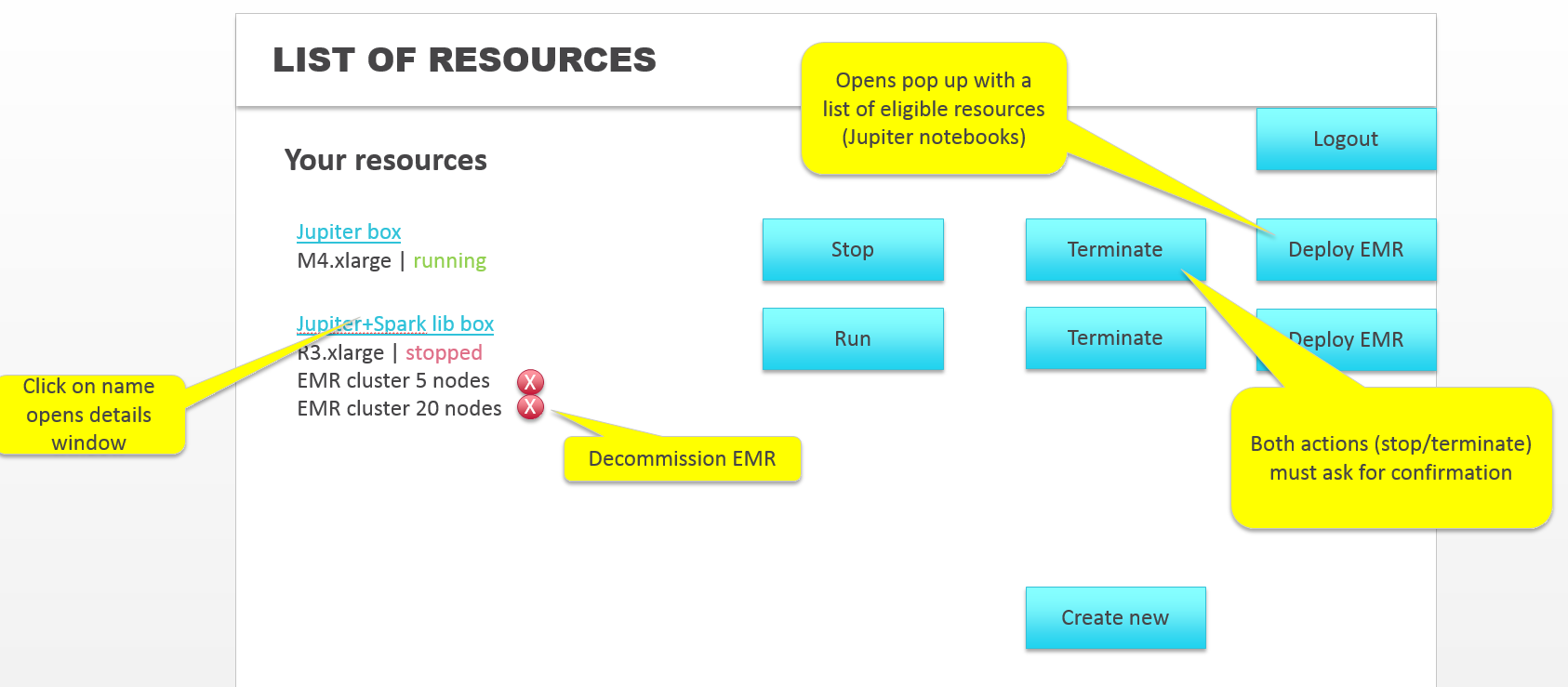


Figure 3 List resources

Also, the page Figure 3 List resources is display to user with preconfigured env (subnet, storage, etc, so user is not the first time here and already registered vital components). On this page user can see short description of running resources like resource name, resource size (shape for EC2 instance and number of slaves X shape of slave for EMR) and status of resource (which can be *running* or *stopped*). There are 2 actions possible on these resources:

* Terminate on any listed resource which must terminate instance
* Stop on running company or Start on stopped instance

If analytical node has attached EMR clusters, it’s showed under the node and each of them can be terminated independently.

At the bottom, there is *Create new* button which leads to new page or popup with an information required to create a new analytical tool and depictured on Figure 4 Create new analytical tool

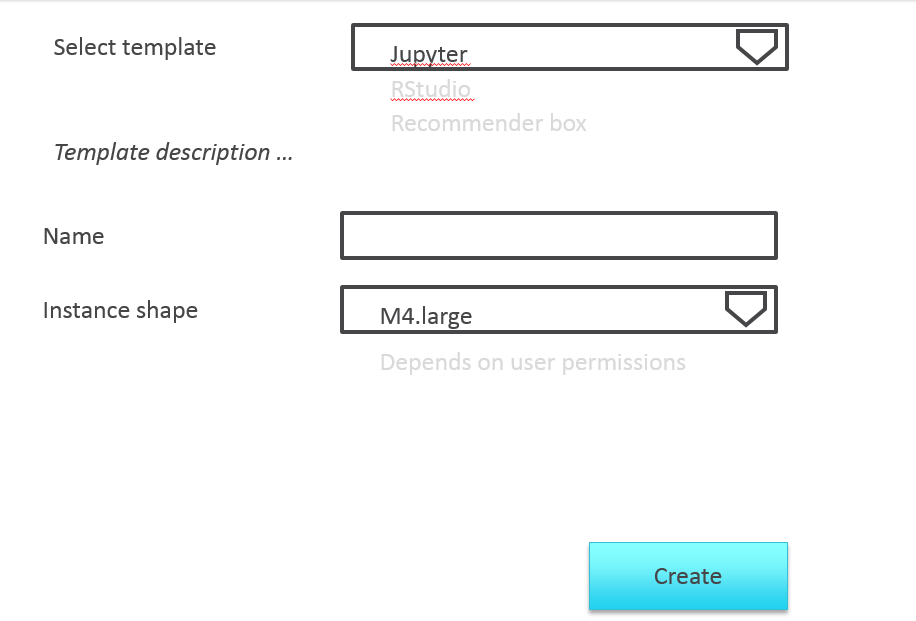


Figure 4 Create new analytical tool

Back to Figure 3 List resources, each analytical node has a button Deploy EMR which allows to create and attached EMR with specific number of nodes to this service as shown in expected popup Figure 5Create computational resources

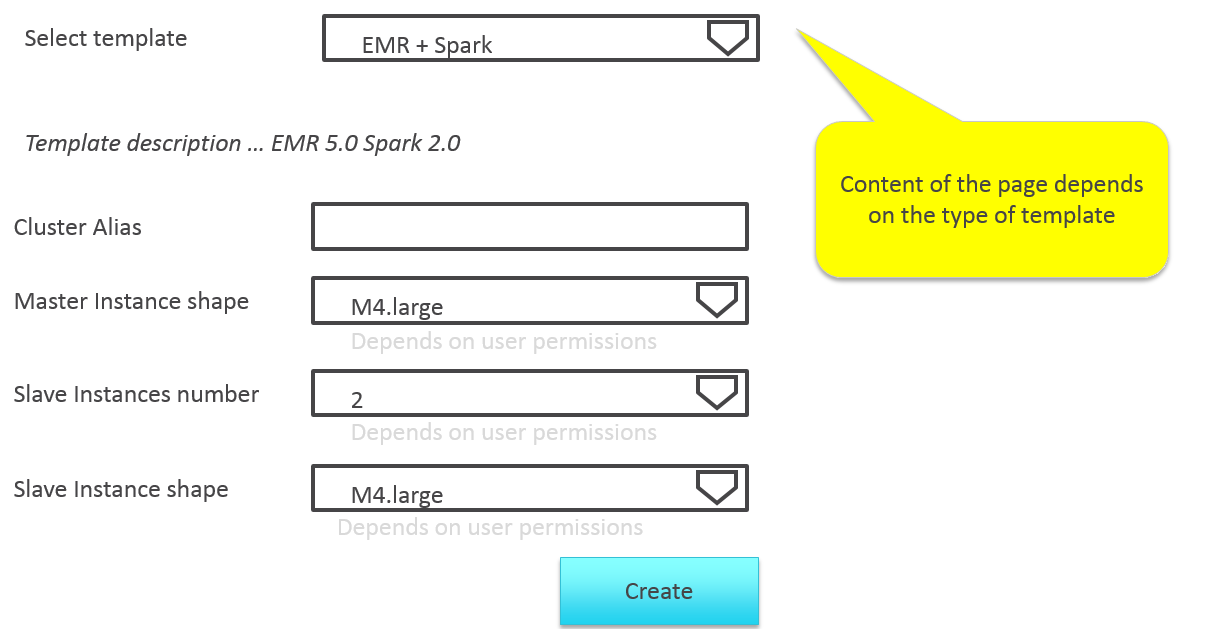


Figure 5Create computational resources

Click on each resource listed on Figure 3 List resources must show details about box and instruction how to connect to this node, as shown on Figure 6 Details

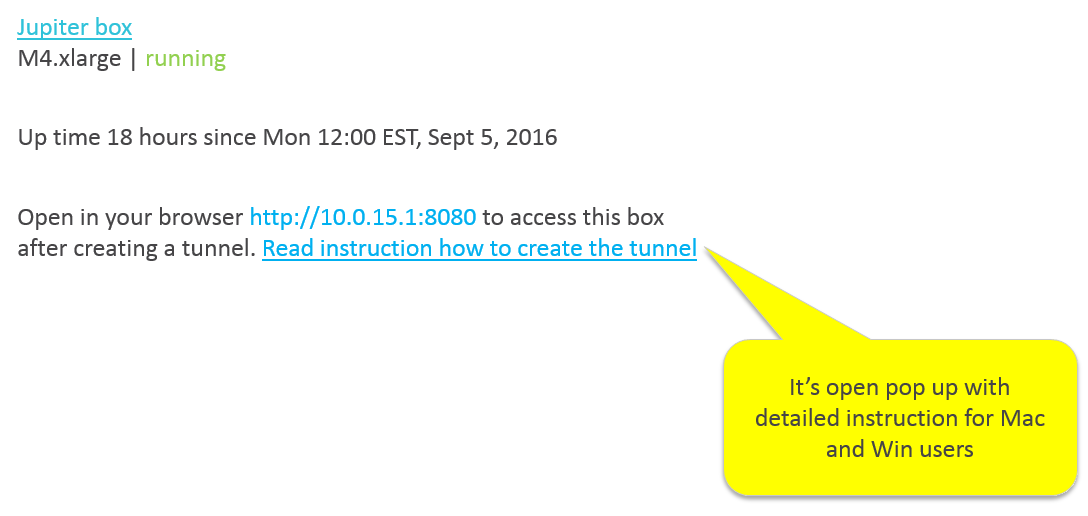


Figure 6 Details

## Non-functional requirements

[**Description:** This section lists business and technology drivers, motivation on the customer side (from the customer’s and Solution Architect’s points of view)

**Section Type:** Mandatory (if section is not covered with available documentation)]

### Self service: UI

There are two options: build (1) simple Web UI under MVC architecture where web application returns server-side generate web page or build (2) SPA backed by a set of RESTful web service.

The benefits of (2) is an implemented REST API which can be used as integration point with 3rd party systems. At the same time, there is the risk that such kind of application is significantly more difficult to implement and more expensive to support.

Both approaches are expected to be validate and the best one picked after checking available resources and cost/benefit estimation.

## Constraints

[**Description:** This section lists constraints and explanations for them. This is critical to do in order to put all constraints together and repeat them for the customer and the Solution Architect dealing with this particular solution.

**Section Type:** Mandatory (if section is not covered with available documentation)]

## Assumptions

[**Description:** This section lists all the assumptions made by the Solution Architect with explanations for them. This is critical to do in order to cover all the gaps in the requirements.

**Section Type:** Mandatory (if section is not covered with available documentation)]logy mapping for components

1. **Initial setup docker image** – docker image responsible for infrastructure initial setup.
   * Initial setup docker image shall be pre-built and stored in EPAM docker registry.
   * Initial setup docker image shall be infrastructure agnostic and shall support (Azure, AWS, GCP, VMWare)
   * Initial setup docker image components:
     + Fabric, Boto scripts
     + Python
     + Environment configuration templates (.py, .fab, .boto)
   * Initial setup docker image shall be based on 14.04 LTS
   * Initial setup docker image shall additionally contain CLI for particular provider (AWS, Azure, GCP)
2. **Deployment host** – due to various differences in platforms (Windows, Linux, MacOS) a separate linux host shall be pre-configured in the cloud for infrastructure deployment initiation.
3. **Management server** – single host for infrastructure management and provisioning. Administrators and data scientists shall use this server for creating/management of exploratory environment. Management server components

* Management docker image – image that contains scripts, templates for infrastructure management.
  + - * Based on Ubuntu 16.04 LTS.
      * Boto, fabric, python scripts for infrastructure management.
      * Infrastructure templates.
      * Configuration database.
* **Management server UI** – web based UI with responsive design for infrastructure management.

1. **Exploratory environment** – data science environment created per data scientist. Consists of:

* Analysis server – server where analytics tools shall be configured for each scientist:
  + Data Exploration – Jupyter.
  + Local computation resources - local Apache Spark with python and scala kernels.
* Computation resources – cloud-based computation resources (EMR, HDInsight)
* Storage – per each data scientist separate storage resources shall be provisioned.

1. **Collaboration space** – resources for sharing data between data scientists and exploratory environments.

|  |  |  |
| --- | --- | --- |
| Component | Technology | Description |
| Server OS | Ubuntu 16.04 LTS |  |
| COntainer tool | Docker |  |
| Scripting | Boto, python |  |
| Web Server | Tomcat / Jetty |  |
| WEB UI frameworks | React, Immutablejs / Angular, Bootstap, jquery |  |
| Web ui Backend | Java 8, Dropwizard / Nodejs / Pypy, TOrnado |  |
| Configuration db | HBase, Redis |  |
| Unit testing | js – Jest, java – mockito, junit |  |
| Build tool | maven / gradle |  |

# Quality Attributes

## Considered Solution Architecture Quality Attributes

The risk and non-risk issues identified in this assessment are determined by evaluating how well the software architecture meets the [*architectural quality attributes*](http://msdn.microsoft.com/en-us/library/ee658094.aspx) proposed by [*Microsoft in the Microsoft Application Architecture Guide.*](http://msdn.microsoft.com/en-us/library/ff650706.aspx) See quality attributes in “Appendix A: Solution Architecture Quality Attributes“.

[**Description:** List all the quality attributes mapped to corresponding main functional and non-functional requirements.

TODO: Propose a set of **all considered** attributes without prioritization.

**Section Type:** Highly recommended]

| Functional requirement | Quality attribute | Comment |
| --- | --- | --- |
|  |  |  |
|  |  |  |

| Non-functional requirement | Quality attribute | Comment |
| --- | --- | --- |
|  |  |  |
|  |  |  |

## Architecturally Significant Quality Attributes

[**Description:** Select 3-5 most important quality attributes for the future solution architecture based on the requirements because your architectural decisions will depend on them. Provide motivation for selecting every quality attribute.

List attributes for both the baseline architecture and the target architecture.

Define a set of measurable metrics for each quality attribute. These metrics will show whether a particular quality attribute is achieved or not. The components where the metric is measured should also be noted. Fill in the table below.

TODO: Propose a set of ready-to-use attributes and metrics for them.

Section Type: Mandatory]

| Quality attribute | Measurable metric | Related components |
| --- | --- | --- |
|  |  |  |
|  |  |  |

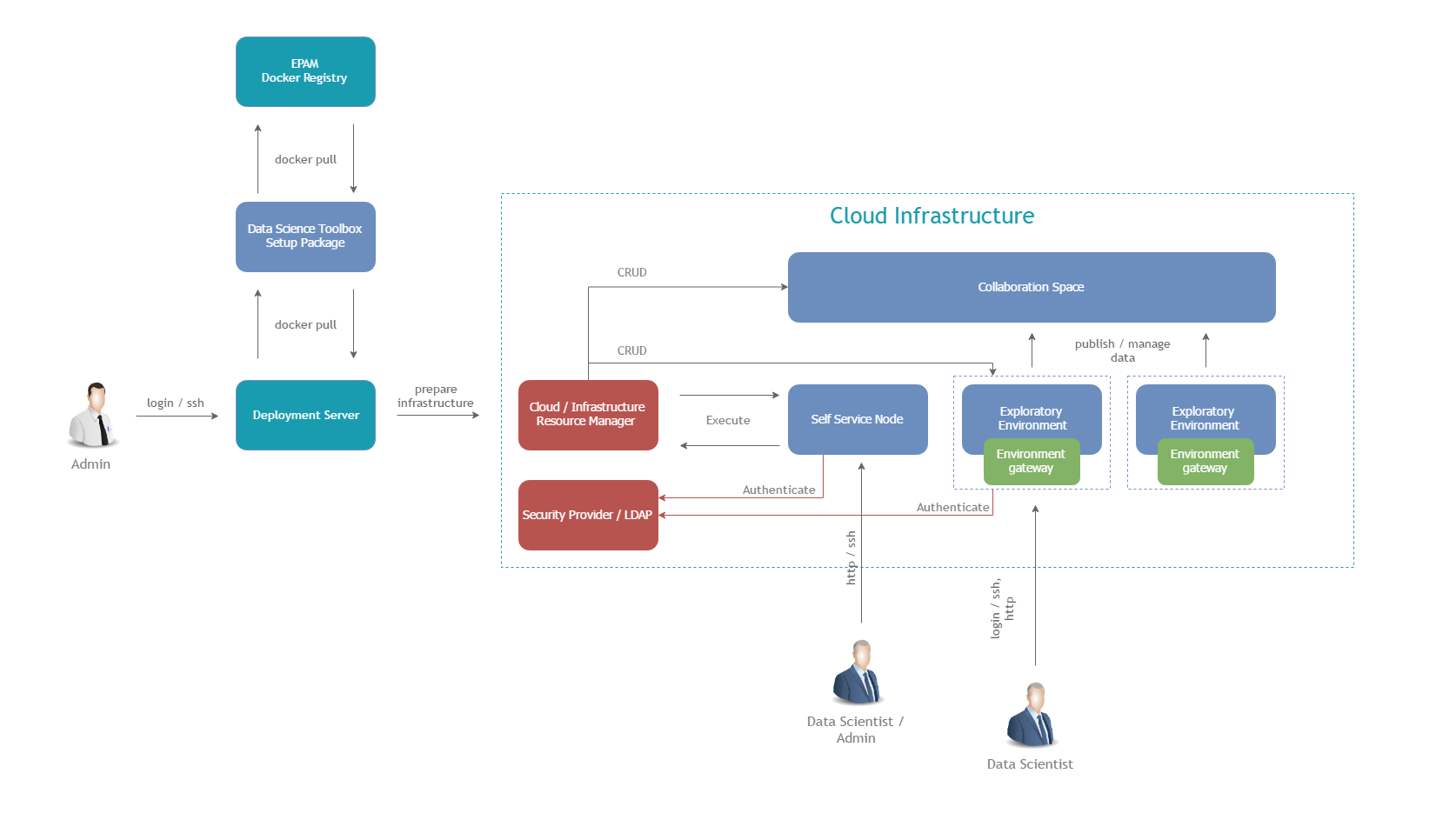
# Baseline Solution architecture

[**Description:** This section must be addressed only if you are working in a “brown field”, where the customer already has a legacy solution. This section describes the legacy solution architecture with a sufficient level of detail.

Note: In some cases it makes sense to have Baseline Solution Architecture (also known as putting the solution architecture “AS IS” in a separate document.

**Section Type:** Mandatory (if legacy solution is to be re-worked and migrated)]

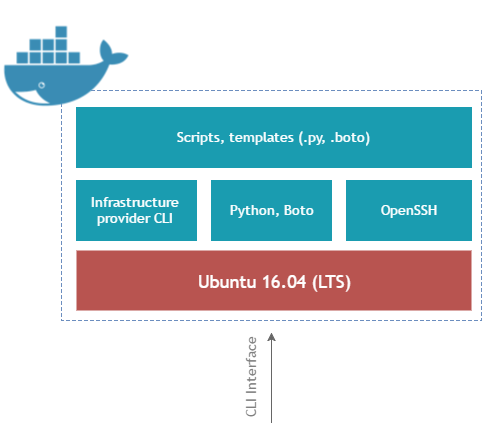
## High-level solution structure



### DEPLOYMENT SERVER

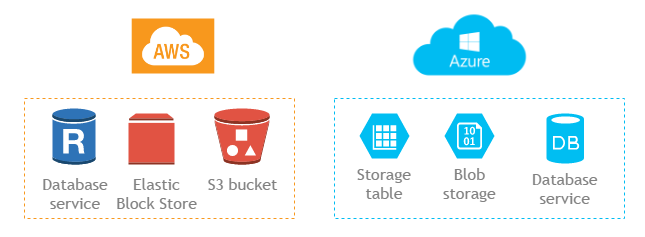
|  |  |
| --- | --- |
| **Description** | In order to minimize efforts required for initial infrastructure setup and resolve issues related to docker and OS differences (Linux, Windows, MacOS) an pre-configured image shall be prepared.  Additionally, python/boto script shall be implemented that creates machine from scratch. |
| **Technology Stack** | Following software shall be installed in deployment server:   * Ubuntu 16.04 LTS. * OpenSSH. * Git, additionally access to EPAM Git repository shall be configured. * Docker, additionally access to EPAM docker registry shall be configured. |
| **Related components** | [List related components with a short description of the relation nature] |
| **Covered functional requirements** | [List covered functional requirements] |
| **Notes** | [Put any additional specific notes here] |

### INITIAL SETUP PACKAGE



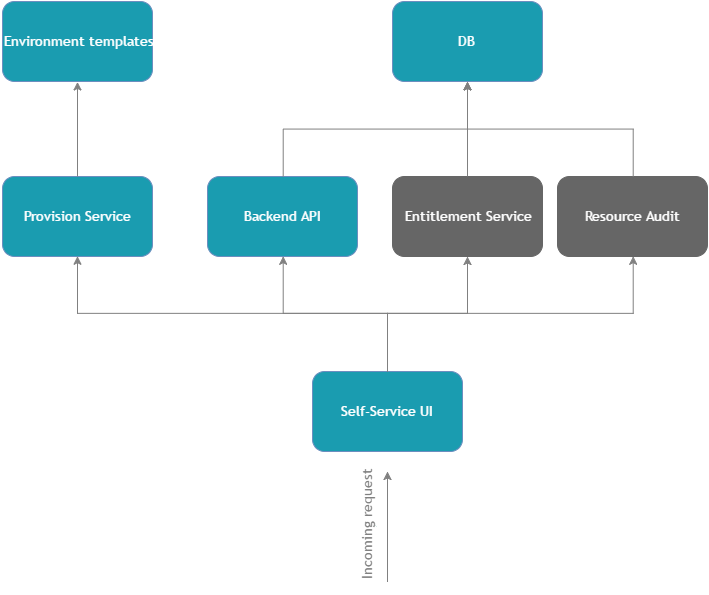
|  |  |
| --- | --- |
| **Description** | Initial setup package is prepared docker container which is stored in EPAM docker registry. |
| **Technology Stack** | Initial setup package shall be based on Ubuntu 16.04 image with following additional components installed:   * Infrastructure provider CLI – in order to reduce additional dependency installation and simplify configuration and communication with Infrastructure Resource Manager (AWS SLI or Azure SLI for example). * Python, Boto shall be installed in order to run .py, .boto scripts. * Scripts and Templates for infrastructure setup. |
| **Related components** | [List related components with a short description of the relation nature] |
| **Covered functional requirements** | The main functionality of initial setup package is:   * Perform initial infrastructure configuration. * Simplify deployment and setup scenarios.   Initial setup package shall configure following components in the cloud:   * Cloud resource group. (For AWS Tagging shall be used to mark all components as resource group) * Cloud vnet. * Create management node subnet. * Create and configure Self-Service node. (For AWS subnet for self-service node, ec2 instance for self-service node * Create and configure Collaboration space. (For AWS S3 bucket with pre-configured security policies) * Security group and profiles (For AWS iam profile) |
| **Notes** | [Put any additional specific notes here] |

### COLLABORATION SPACE (NOT IN SCOPE OF MVP



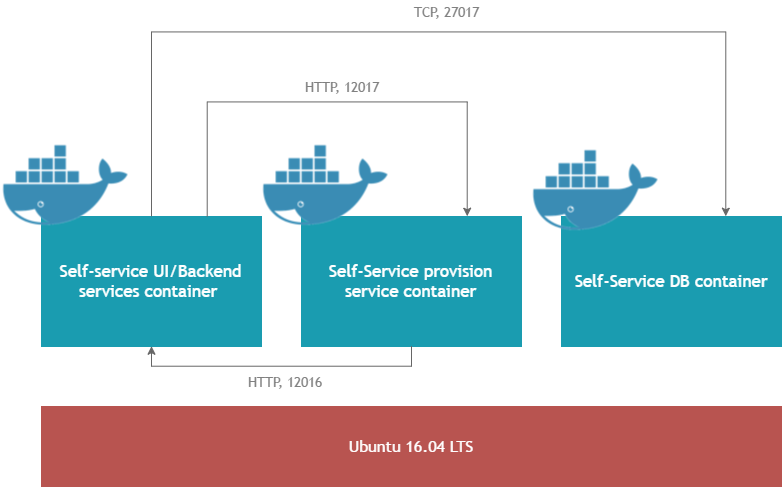
|  |  |
| --- | --- |
| **Description** | Collaboration space shall be deployed in infrastructure subnet accessible from Exploratory |
| **Technology Stack** | For now, Azure and AWS is supported. In Azure collaborative space shall be implemented as:   * Storage Table. * WASB Storage. * Database Service.   In AWS collaborative space shall be implemented as:   * S3 buckets. * Database Service. * Elastic Block Storage. |
| **Related components** | [List related components with a short description of the relation nature] |
| **Covered functional requirements** | Environment and shall serve as collaborative data storage where data scientists can copy and share their data (data, metadata, notebooks). |
|  |  |
| **Notes** | [Put any additional specific notes here] |

### SELF-SERVICE NODE



NOTE: Components marked as gray are out-of-scope.

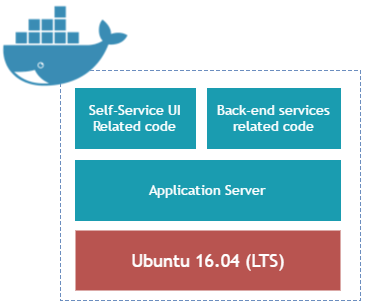
**Physical schema**



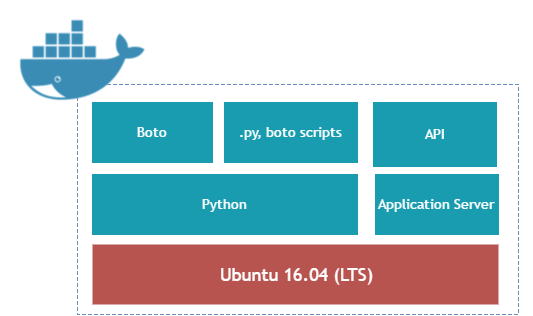
|  |  |
| --- | --- |
| **Description** | Self-service node major components:   * Self-service node web UI. * Self-service node DB. * Self-service node provisioning service. * Self-service node entitlement manager. * Environment templates. * Resource audit. |
| **Technology Stack** |  |
| **Related components** | [List related components with a short description of the relation nature] |
| **Covered functional requirements** | Self-service node shall serve as management component responsible for:   * Exploratory environment configuration and management (CRUD). * Collaboration space configuration. * User mapping to resource policy. |
| **Notes** | [Put any additional specific notes here] |

It is recommended to implement 3 containers for self-service components.

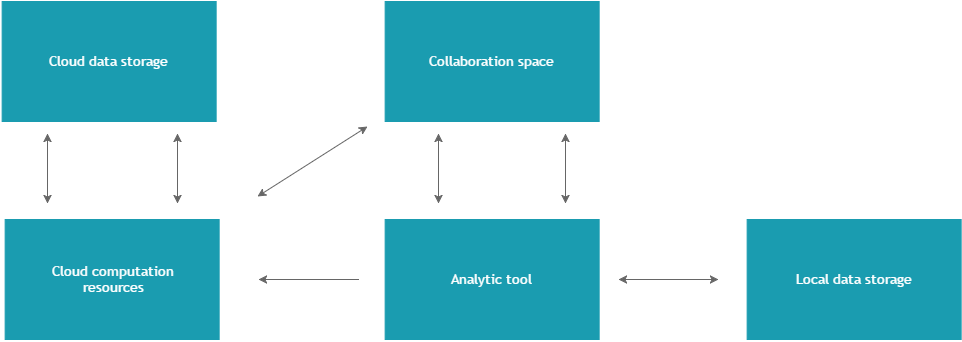
1. Self-service UI/Backend container – this container shall consist of UI and backend services.



1. Self-service DB container – mongoDB docker container.
2. Provision service container – container responsible for infrastructure management.

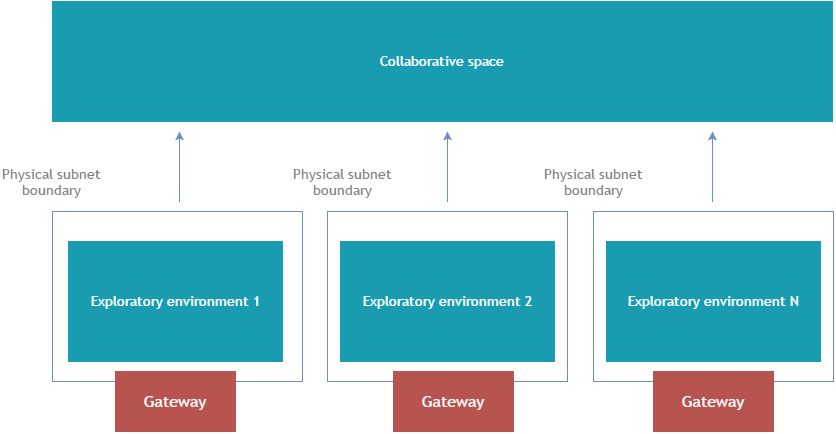


### EXPLORATORY ENVIRONMENT

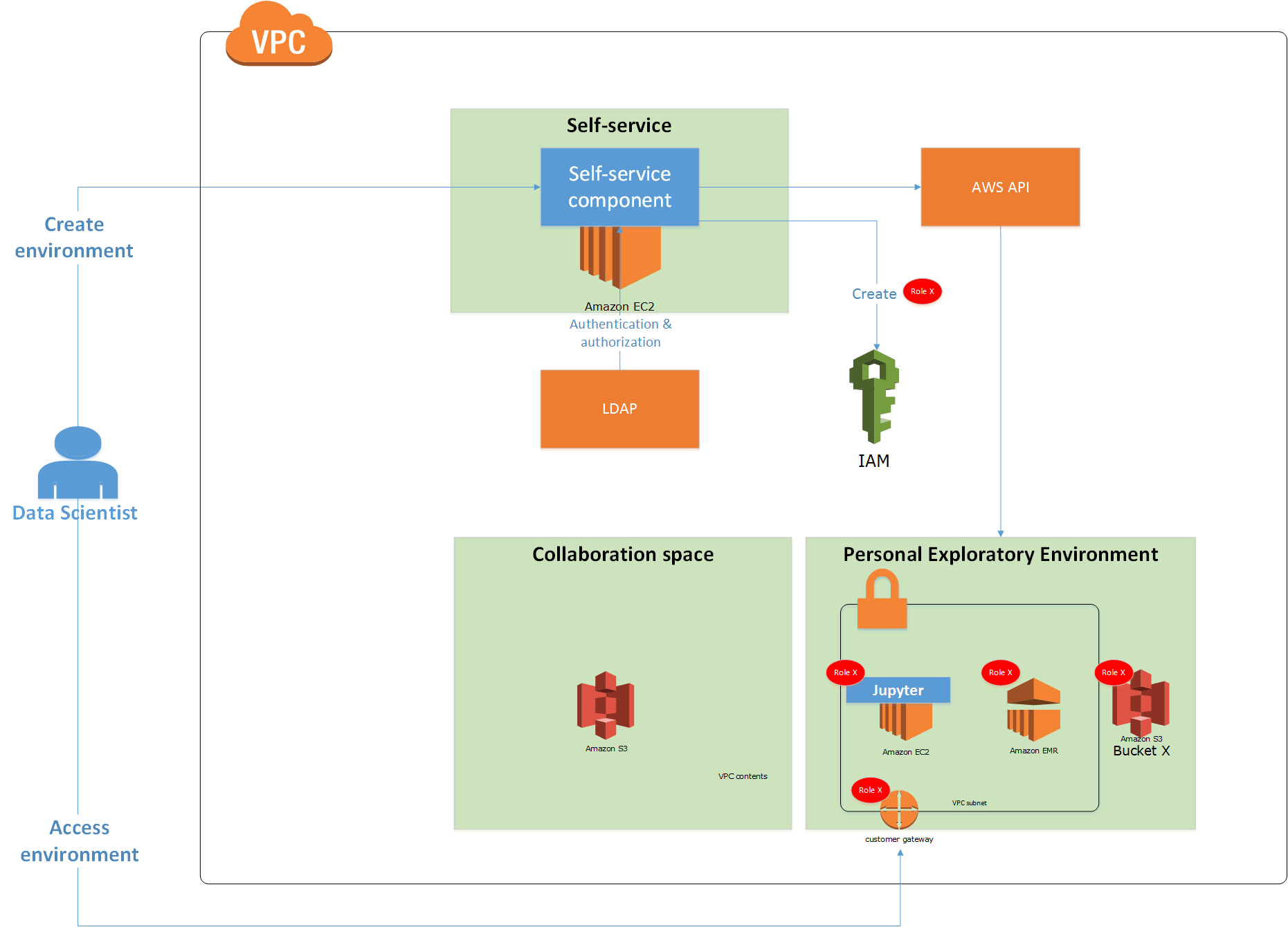


|  |  |
| --- | --- |
| **Description** | Exploratory environment consists of following components: |
| **Technology Stack** | * Analytic tool   + Jupyter with python and scala kernels.   + MLLib, Wisp, Breeze, MatPlotLib libraries configured in Jupyter. * Local computation kernel – local version of Spark kernel configured on same physical server as Analytic tool * Cloud computation resource – cloud-based Spark cluster provided by Amazon (EMR) or Azure (HDInsight) * Data storage – per each exploratory environment separate storage resources shall be created. |
| **Related components** | [List related components with a short description of the relation nature] |
| **Covered functional requirements** |  |
|  |  |
| **Notes** | [Put any additional specific notes here] |

Each exploratory environment shall be isolated and put in separate subnet which will not be accessible from other exploratory environments, however access to collaborative space shall be available.



## SOLUTION IN AWS CLOUD



**SOLUTION IN MS AZURE CLOUD**

### 

## Solution components structure

[**Description:** This section describes the internal structure of architecturally significant components.

For every solution component, describe the following things:

* Diagram of sub-components
* Description of the sub-components’ behavior (in terms of sequences, state-machine, data flow and execution diagrams)
* Sub-component integration

**Section Type:** Mandatory (if legacy solution is to be re-worked and migrated)]

### SELF-SERVICE UI COMPONENT

Web Application that shall provide users with following functionality depending on user role:

* Exploratory environment CRUD
  + Data scientist can manage only his own environments

Self-service UI implementation options:

1. Fluid design SPA application based on FLUX architecture with REST API backend services (recommended)
   1. Front-end technologies – ReactJS, ImmutableObjectJS, Facebook Dispatcher, RequireJS, jQuery, Bootstrap, jest.
   2. Backend:
      1. Java – Java 8, Jetty, dropwizard.io, mockito, junit.
2. MVC application with server-side generation of client-side markup and logic
   1. Back-end technologies – Java 8, Jetty, Spring, junit, mockito.

|  |  |  |
| --- | --- | --- |
|  | Fluid design SPA /w REST | Server-generated web |
| Flexibility to modify | Full control over markup and styling. | Limited control, harder to implement and attach nice views. |
| Scalability / Portability | Due to service design much easier to scale and re-use functionality. | Tightly coupled design. |
| Maintainability | Separated UI, granular backend make system easier to test, support and troubleshoot. | More efforts required for system support and testing. |
| Future | Industry standard. | This approach is no longer treated as industry standard. |
| Costs | About 2x-3x higher cost for UI implementation than server generated web approach. |  |

[Describe main sub-components in this section:

* Sub-component contract
* Sub-component diagram
* Sub-component relations/dependencies]

### PROVISION SERVICE

Service responsible for infrastructure management:

* Exploratory environment CRUD.
* ~~Collaborative space management.~~
* Manage infrastructure templates.

Provision service shall provide API for these operations.

Provision service shall be implemented in technology selected in Self-Service UI and run Boto/Python scripts as well environment templates.

### BACKEND

Backend for self-service UI, shall perform various operations as well as communication with database.

### Authentication and Authorization approach

For MVP scope only support of LDAP is planned to be implemented. The goal is to achieve flexible and pluggable solution which allows to replace different identity provider with a minimal effort. LDAP is the primary goal for MVP and the following solution is proposed to solve compatibility between customer LDAP schema and solution entities:

Enterprise

AD

with LDAP interface

Request

LDAP integration

Mapping component

Self-service backend

Self-service UI

Figure 7 LDAP integration

The idea of mapping component is to provide flexible approach to map attributes from customer’s LDAP to solution own entities. For example, each customer can have different attribute which specify user’s group. The mapping component is implement as Python file (in fact, Jython) which is executed in JVM and perform a mapping of LDAP attributes to general model. Mentioned Python file can be easy edited and replaced without a need to restart server.

The example of mapping is depictured on the diagram Figure 8 Mapping using Python

**Java LDAP connector**

*Java class*

(&(givenName=John)(l=Dali))

**LDAP**

email

given name

last name

**MAPPING**

*Python code*

email => email

given name + last name => full name

**USER**

*Java bean*

email

full name

Figure 8 Mapping using Python

### DB

Management node DB shall contain following information:

* Exploratory environment information.
* Collaborative environment information.
* Environment audit information.
* It is recommended to consider storing templates in database.

****

For database technology MongoDB is recommended as it supports large variety of frameworks and technologies as is document database, also it fits well for storing json configuraiton.

## Domain model

[**Description:** This section describes the domain model if Domain-Driven Design was used during baseline solution design. Describe the domain model in terms of bounded contexts and the main entities in every context.

**Section Type:** Optional (if applicable)]

## Data model

[**Description:** This section describes the approach to data storage and data models. List data storage components of any nature, relational and non-relational, with data model diagrams for each storage.

**Section Type:** Highly recommended (if applicable)]

### Storage 1

[This section describes the data storage technology stack and the used data model with data model diagrams.]

### Storage 2

[This section describes the data storage technology stack and the used data model with data model diagrams.]

### …

## High-level deployment approach

[**Description:** This section describes the high-level deployment approach to all required environments (development, staging, production, etc.). This is a blueprint in high-level detail without specific physical parameters of hardware and so on.

**Section Type:** Mandatory (if legacy solution is to be re-worked and migrated)]

## Architecturally Significant Quality Attributes

### Security

[**Description:** Possible focus areas to consider in this section:

* Customer security policies
* Authorization/authentication
* Communication encryption
* Encryption of stored data
* Personal data management
* Deployment security
* Etc.

**Section Type:** Recommended (if legacy solution is to be re-worked and migrated)]

### Supportability

#### Monitoring

[**Description:** Describe the approach to solution support here. Consider the following aspects:

* Logging
* Performance counters/metrics
* Monitoring tools
* Solution component availability
* Etc.

**Section Type:** Recommended (if legacy solution is to be re-worked and migrated)]

#### Administration Tools

#### Specific deployment aspects

[**Description:** For example, complex database deployment]

[Notice: Note that you must consider the migration process, risks, issues, and so on if you have a legacy solution to migrate to the new architecture. In this case, you need to design the new solution having this legacy architecture in mind.

Section Type: Optional]

# Target Solution Architecture

[**Description:** This section describes the proposed target solution architecture (also known as solution architecture “to be”).

Section Type: Mandatory]

## Risks

[**Description:** This section lists potential risks related to the target solution architecture or solution migration to target solution architecture.

**Section Type:** Highly recommended]

## dependencies

[**Description:** This section lists dependencies related to the target solution architecture or solution migration to target solution architecture.

**Section Type:** Highly recommended]

## Assumptions

[**Description:** This section lists assumptions related to the target solution architecture or solution migration to target solution architecture.

**Section Type:** Highly recommended]

## High-level solution structure

[**Description:** This section describes (in the context of the whole IT landscape):

* Set of solution components
* Solution component integration
* High-level architectural styles to be used in the solution architecture

Include the following things:

* High-level solution diagram
* High-level description of the solution’s behavior (in terms of sequences, state-machine, data flow and execution diagrams)
* Comments about architectural styles to be used
* List of architecturally significant components and descriptions of their technology stack and integration with each other

Section Type: Mandatory]

### Solution Component 1

|  |  |
| --- | --- |
| **Description** | [Describe the general purpose of the component in the system] |
| **Technology Stack** | [Describe the technology stack of the component by listing the main frameworks, libraries, tools, etc.] |
| **Related components** | [List related components with short descriptions of the relation’s nature] |
| **Covered functional requirements** | [List covered functional requirements] |
| **Notes** | [Put any additional specific notes here] |

### Solution Component 2

|  |  |
| --- | --- |
| **Description** | [Describe the general purpose of the component in the system] |
| **Technology Stack** | [Describe the technology stack of the component by listing the main frameworks, libraries, tools, etc.] |
| **Related components** | [List related components with short descriptions of the relation’s nature] |
| **Covered functional requirements** | [List covered functional requirements] |
| **Notes** | [Put any additional specific notes here] |

### …

## Solution components structure

[**Description:** This section describes the internal structure of architecturally significant components.

For every solution component, describe the following:

* Diagram of sub-components
* Description of the sub-components’ behavior (in terms of sequences, state-machine, data flow and execution diagrams)
* Sub-component integration

**Section Type:** Highly recommended (to describe solution components in details)]

### Solution Component 1

[**Description:** Describe the main sub-components in this section:

* Sub-component contract
* Sub-component diagram
* Sub-component relations/dependencies

**Section Type:** Highly recommended (to describe solution component in technological details)]

### Solution Component 2

[**Description:** Describe the main sub-components in this section:

* Sub-component contract
* Sub-component diagram
* Sub-component relations/dependencies

**Section Type:** Highly recommended (to describe solution component in technological details)]

### …

## Domain model

[**Description:** This section describes the domain model if Domain-Driven Design was used during baseline solution design. Describe the domain model in terms of bounded contexts and the main entities in every context.

**Section Type:** Highly recommended (If applicable, in order to direct the development to Domain-Driven Design practices)]

## Data model

[**Description:** This section describes the approach to data storage and data models. List data storage components of any nature, relational and non-relational, with data model diagrams for each storage.

**Section Type:** Highly recommended (If applicable, in order to form approach to data storing)]

### Storage 1

[This section describes the data storage technology stack and the used data model with data model diagrams.]

### Storage 2

[This section describes the data storage technology stack and the used data model with data model diagrams.]

### …

# Technology stack

|  |  |  |
| --- | --- | --- |
| Technology | Usage | Description |
| Ubuntu 16.04 x64 LTS | OS image and version used as base image / OS for Edge, Self-Service, Exploratory Environment Nodes, Docker base image | One of the leading open-source OS platforms. |
| Docker | Containerization and isolation platform used as container for Initial Setup Package, Self Service Container | Open-source web-based platform for data-analytics. |
| MongoDB | Database for Self-Service node. |  |
| Jupyter 4.1 | Used in exploratory environment as workplace for data scientist. | Open-source web-based platform for data-analytics. |
| Apache Spark 1.6.0 (hdInsight, Notebook Server) | Data processing engine used in Exploratory environment. | Open-source Big Data batch in-memory data processing platform. |
| Anaconda (Python 2.7) | Used in exploratory environment in Jupyter. | Open-source Data Science Platform |
| ~~nGinx~~ 1.11.1 Community Edition | Reverse Proxy for Data Science Toolbox. | Open-source reverse proxy. |
| MLLib 1.5.1 | Visualization library for Jupyter. | OpenSource python-based visualization library. |
| PANDAS 0.18.1. | Data analysis toolkit. | OpenSource python-based analysis toolkit for working with relational and labeled data. |
| Scala Kernel for Jupyter | Used in exploratory environment for implementation of data science scripts in Scala. | Functional language based on JVM. |
| Breeze, Wisp | Scala-based visualization frameworks for Jupyter. |  |
| Python, Boto | Environment preparation scripts. |  |
| ReactJS, ImmutableObjectJS, Dispatcher | Front-end libraries for SPA development. | One of the leading Javascript frameworks for client-side UI implementation based on Flux architecture |
| jQuery, requirejs | Helper frameworks for Javascript. |  |
| Bootstrap | Helper framework for front-end development and styling. | One of the most popular frameworks for UI development. |
| Java 8, dropwizard.io, NodeJS, Sails.js, Waterline | Backend frameworks for Self-Service UI |  |
| Jetty | HTTP server based on JVM |  |

## High-level deployment approach

[**Description:** This section describes the high-level deployment approach to all required environments (development, staging, production, etc.). This is a blueprint in high-level detail without specific physical parameters of hardware and so on.

Section Type: Mandatory]

## Architecturally Significant Quality Attributes

### Security

[**Description:** Possible focus areas to consider in this section:

* Customer security policies
* Authorization/authentications
* Communication encryption
* Encryption of stored data
* Personal data management
* Deployment security
* Etc.

**Section Type:** Mandatory (If this quality attribute is architecturally significant one)]

### Supportability

[**Description:** Address supportability.

**Section Type:** Mandatory (If this quality attribute is architecturally significant one)]

#### Monitoring

[**Description:** Describe the approach to solution support. Consider the following aspects:

* Logging
* Performance counters/metrics
* Monitoring tools
* Solution component availability
* Etc.

**Section Type:** Mandatory (If this quality attribute is architecturally significant one)]

#### Administration Tools

#### Specific deployment aspects

[**Description:** For example, complex database deployment]

[Notice: Note that you must consider the migration process, risks, issues, and so on if you have a legacy solution to migrate to the new architecture. In this case, you need to design the new solution having this legacy architecture in mind.

Section Type: Optional]

## Architecture Constraints and Limitations

[**Description:** This section describes all the constraints and limitations that the proposed solution architecture has.

The Customer must be made aware of them, so you must communicate them explicitly and clearly.

Section Type: Optional]

# Transition

[**Description:** This section describes the approach to migration and compatibility (applicable in the case of “brown field”, when you need to migrate the customer smoothly from a legacy solution to a new one, including customer data.

It is recommended to provide the customer with a set of high-level transition architectures (if applicable).

Also, it is recommended to describe:

* General approach to backward/forward compatibility (if applicable)
* General approach to data compatibility
* General approach to interface compatibility
* Component-by-component migration description
* Migration scenario and data migration should be described (physical view)
* Deployment windows and how deployment affects availability
* Level of support needed from others (operations, 3rd-party vendors, etc.)
* Etc.

**Section Type:** Mandatory (if migration from legacy solution must be done)]

1. Solution Architecture Quality Attributes

See considered solution architecture quality attributes in the table below:

|  |  |  |
| --- | --- | --- |
| Category | Quality attribute | Description |
| Design Qualities | Conceptual Integrity | Conceptual integrity defines the consistency and coherence of the overall design. This includes the way that components or modules are designed, as well as such factors as the coding style and variable naming. |
| Maintainability | Maintainability is the ability of the system to undergo changes with a degree of ease. These changes can impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements. |
| Reusability | Reusability defines the capability for components and subsystems to be suitable for use in other applications and in other scenarios. Reusability minimizes the duplication of components, and the implementation time. |
| Run-time Qualities | Availability | Availability defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. Availability is affected by system errors, infrastructure problems, malicious attacks, and system load. |
| Interoperability | Interoperability is the ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties. An interoperable system makes it easier to exchange and reuse information internally as well as externally. |
| Manageability | Manageability defines how easy it is for system administrators to manage the application, usually through sufficient and useful instrumentation exposed for use in monitoring systems and for debugging and performance tuning. |
| Performance | Performance is an indication of the responsiveness of a system to execute any action within a given time interval. It can be measured in terms of latency or throughput. Latency is the time required to respond to any event. Throughput is the number of events that take place within a given amount of time. |
| Reliability | Reliability is the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval. |
| Scalability | Scalability is ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged. |
| Security | Security is the capability of a system to prevent malicious or accidental actions outside of the designed usage, and to prevent disclosure or loss of information. A secure system aims to protect assets and prevent unauthorized modification of information. |
| System Qualities | Supportability | Supportability is the ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly. |
| Testability | Testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. Good testability makes it more likely that faults in a system can be isolated in a timely and effective manner. |
| User Qualities | Usability | Usability defines how well the application meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, providing good access for disabled users, and resulting in a good overall user experience. |