SDP Platform Services Component and Connector View

Document number……………………………………………………………	SKA-TEL-SDP-0000013
Document Type…………………………………………………………………………………...	REP
Revision………………………………………………………………………………………………	05
Author………………………	J. Garbutt, J. Taylor, P. Harding, A. Ensor, V. Allan, P. Wortmann
Release Date…………………………………………………………………………	2018-04-23
Document Classification…………………………………………………………………	Unrestricted
Status…………………………………………………………………………………...	Released

Lead Author	**Designation**	**Affiliation**
John Garbutt	SDP Team	StackHPC
Signature & Date:	John Garbutt John Garbutt (Apr 25, 2018)	

Released by	**Designation**	**Affiliation**
Paul Alexander	SDP Project Lead	University of Cambridge
Signature & Date:	Paul Alexander Paul Alexander (Apr 25, 2018)	

Table of Contents

List of Abbreviations

AAAI	Authorization, Access, Authentication and Identification
API	Application Programming Interface
AZ	Availability Zone
C&C	Component and Connector
CPU	Central Processing Unit
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
IPMI	Intelligent Peripheral Management Interface
LAN	Local Area Network
NVME	Non-volatile Memory Express
PFS	Parallel File System
REST	Representational State Transfer Technology
SAFe	Scaled Agile Framework
SAML	Security Assertion Markup Language
SATA	Serial ATA
SDP	Science Data Processor
SKA	Square Kilometre Array
SSD	Solid State Disk
SSH	Secure Shell
VLAN	Virtual LAN

1. Primary Representation

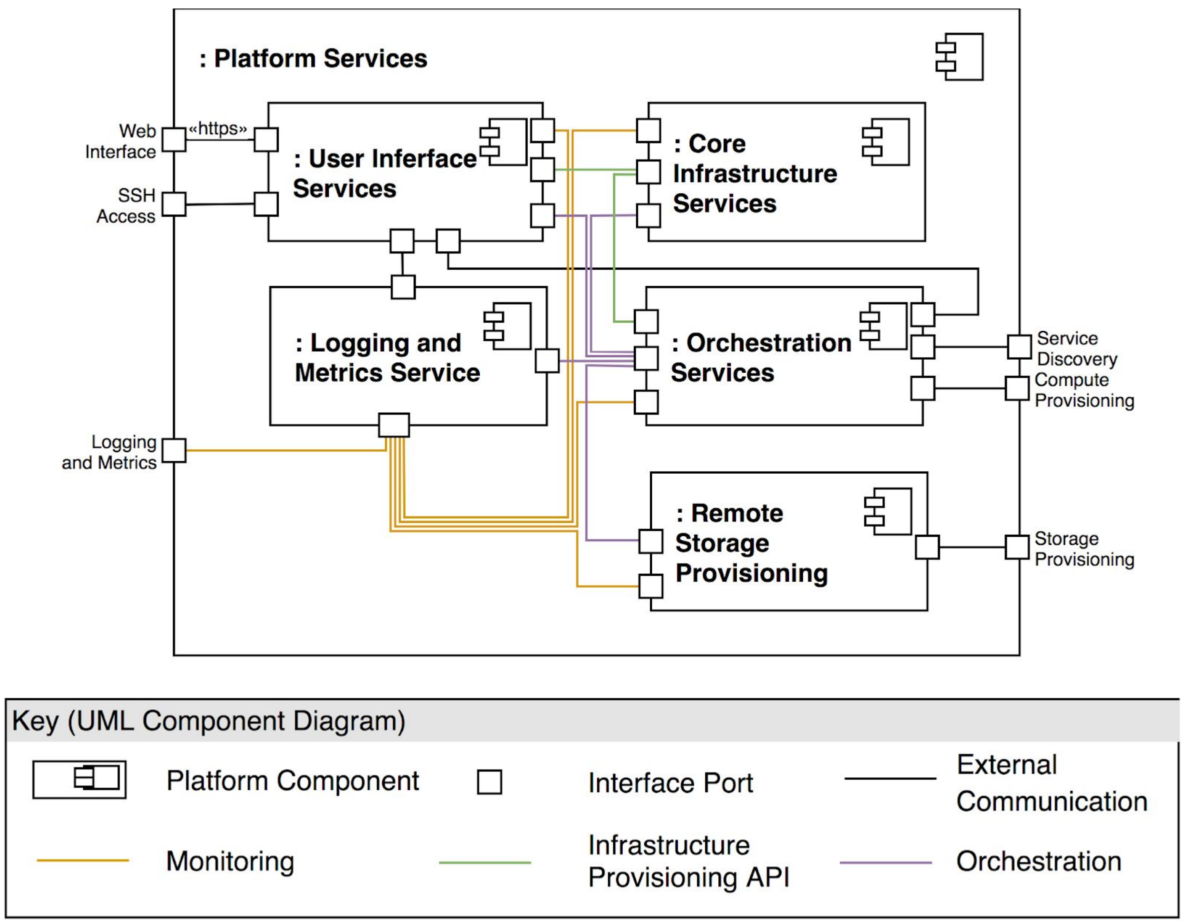

Figure 1: Platform Services Component and Connector Primary Representation

Platform Services are responsible for starting and maintaining the SDP Operational System components [RD01]. The SDP Operational System is successfully started when the TANGO interfaces are available for the Telescope Manager to interact with SDP Operational System. In particular, this includes the provisioning of Compute, Storage and Networking resources and running various services needed by the SDP Operational System.

The Master Controller and Processing Controller of the SDP Operational System are responsible for using Storage Provisioning and Compute Provisioning interfaces provided by the Orchestration Services to deploy the components required for SDP services and processing. The Orchestration Services use the Infrastructure Provisioning API provided by Core Infrastructure Services to manage the hardware such that the Storage, Networking and Compute resources are correctly provisioned. It will then start the appropriate software on these resources, which might be either custom SDP operational system code or standard components provided by Platform Services, such as Databases or Queues. The information needed to connect to these services is communicated to the appropriate SDP Operational System components via the Service Discovery Interface.

Operators will make use of the User Interface Services to obtain an overview of the current state of SDP Operational System and Platform Services components, and to trigger operations such as starting the minimal viable SDP Operational System. There are options for Operators to use either a high-level Web interface or low-level SSH access.

Logging and Metrics Services are provided to all the SDP Operational System and Platform Services that produce logs or metrics. The aggregated logs and metrics are made available to Execution Control and Telescope Manager components via the User Interface Services to support analysis of problems within the SDP Operational System.

Where possible the components and their interfaces are provided “out of the box” by existing technologies leveraged by Platform Services. As such, the exact nature of all the interfaces will only be known once we downselect on a specific implementation of a component. While this is expected at this stage, there could be significant impact for pervasive interfaces such as Metrics collection (e.g. pull vs push model).

Core Infrastructure Services run on dedicated management infrastructure, but all other Platform Services make use of Core Infrastructure Services to provide the infrastructure they require. One exception is that the SSH access noted in User Interface Services must also be deployed on dedicated management infrastructure (Management Servers discussed in the Hardware Decomposition View [RD03]), as the way to perform the very first initial bootstrap of the whole SDP Operational System.

2. Element Catalogue

2.1. Element and Their Properties

This section is a dictionary where each entry is an element of the Primary Presentation. We give the following properties for every element:

- **Functionality:** Description of the functions implemented by the component. This will be the main text of the sub-sections
- **Candidate Implementations:** List of candidate technology choices to help illustrate the kinds of technologies that could be selected to implement a given service

2.1.1 Orchestration Services

The function of Orchestration Services is to provision and maintain components for the SDP Operational System as well as all Platform Services. This consists of allocating and configuring compute, storage and networking resources from Core Infrastructure Services. Component configuration consists of deploying the required environment (e.g. containers, executables, database services, queuing services, and other dependencies), ensuring connectivity with other components (e.g. join the appropriate networks, mount remote storage), and initiating execution with the desired parameters. This process will be configured using automation scripts under version control. While Platform Services are ultimately responsible for curating all the automation scripts, it is expected some automation scripts will be maintained by modules outside of Platform Services.

Orchestration Services maintain a persistent platform configuration, and works to keep the SDP Operational System consistent with that desired state. This means when started up, the platform will automatically provision resources and start services to bring up enough SDP Operational System components to boot-strap remaining components. This will have to include at minimum the Master Controller (Execution Control), but might also cover further pre-configured components, like any software required for providing the TANGO control interface to the Telescope Manager. It is plausible that this set might include most non-processing components of the SDP Operational System, in which case the Master Controller may not have a role in actively provisioning components.

The responsibilities of Orchestration Services includes the initial bootstrap of the SDP Operational System. Typically, the initial starting point would be SSH access to run the orchestration that brings up the User Interface Services component that allows users to trigger the full bootstrap of the rest of Platform Services and bring up the SDP Operational System such that Telescope Manager can access it.

The interfaces to Orchestration Services are:

- There is an interface that connects to all Platform Services, as all the Platform Services are orchestrated by automation in Orchestration Services.
- Orchestration services provision the infrastructure needed to run all the services via the Infrastructure Provisioning API provided by the Core Infrastructure Services. Note that Orchestration Services are responsible for starting the Core Infrastructure Services, which must be done before any orchestration tasks that depend on the Provisioning API.
- The User Interface Services have the ability to trigger actions such as bootstrap the system and trigger and update of the SDP Operational System software.
- The Logging and Metrics service is used to record audit logs of all the operations performed by the Orchestration Services.
- Service Discovery allows the SDP Operational System to discover the services that have been set up by the Orchestration Services. For example, the connection strings for the various SDP databases will be created by the Orchestration that maintains the database, and may be saved in the Configuration Database such that the appropriate services can retrieve the required information.
- The Compute Provisioning interface is used by the Master Controller and Processing Controller of the SDP Operational System to provisioning the components for executing both batch and real-time processing workflows.

The requirement [AD01] SDP_REQ-27 is to ensure the ingest pipeline can be reconfigured within a few seconds to start ingesting a new scheduling block. This requirement is specific to the Compute Provisioning and Storage Provisioning APIs, it does not relate to the Infrastructure Provisioning API of the Core Infrastructure Services. We must be able to quickly and efficiently switch between workflows, but that does not imply we are required to reprovision the infrastructure within the required time frames. Prototyping has highlighted that container orchestration technologies may be a good candidate (to ease the portability of the workloads, with few drawbacks compared to other approaches) for the Compute Provisioning interface.

There are a set of shared services, such as Database and Queue services, that will be provisioned and maintained by Orchestration Services, such that they can be shared between multiple SDP Operational System components. These services are all pre-existing software components. It is expected that the set of shared services will be a relatively static set of services, rather than, for example, there being a need to dynamically create new databases as workflows are started and stopped. During Continuous Integration testing, there will be a fresh bootstrap of a new instance of the SDP Operational System that requires creation of databases, which should reuse the orchestration that is currently used for the initial bootstrap of the two production instances of the SDP Operational System. In either case, there is no need for the SDP Operational System to provision databases at runtime. It is expected that SKA level harmonization efforts will help run these services in an efficient way.

2.1.1.1 Implementations

Candidates:	Ansible, Salt, Kubernetes, Docker Swarm

The current prototype work has focused on a combination of Docker Swarm and Ansible to automate the deployment of the SDP Operational Services, although looking at the use of Kubernetes is also planned. In a similar way, this has included the deployment of the required Database and Queue Services that are expected to be maintained as part of Platform Services. For example, our prototype OpenStack installation uses OpenStack Kolla-ansible and Kayobe to automate the provisioning of OpenStack using Docker containers and Ansible.

2.1.2 User Interface Services

The User Interface Services provides operators with:

- SSH access to a management server to perform both the initial bootstrap of the SDP Operational System and any emergency operational activities
- View the current state of Platform Services by visualizing data from the logging and metrics service, including any alerts.
- View the current state of hardware by visualizing the state of the Core Infrastructure Services
- Modify the SDP Operational System by triggering automated operation tasks performed by Orchestration Services

Looking back at User Interface Services in Figure 1, there are lots of interface ports. We will now describe each of these links and what they are used for:

- A Web Interface is exposed to Operators, and it is the main operator interaction with Platform Services
- SSH access is provided to a management server, as described above. It is expected that from that server there will be SSH access to all dedicated hardware and all hardware that has been provisioned by Core Infrastructure Services.
- There is an interface to Core Infrastructure Services Provisioning API that is used by the Web Interface to visualize the current state of the hardware.

- The Logging and Metrics Service is used to monitor that the various User Interface Services are running, in a similar way to any other Platform Service or External Service.
- There is an additional interface to Logging and Metrics service to query the SDP Operational System, in particular it is used by the Web Interface to visualize the logging and metrics information.
- Orchestration Services are used to provision and maintain all the User Interface Services, in a similar way to any other Platform Service.
- There is an additional interface to Orchestration Services to trigger operations such as the initial bootstrap of the SDP Operational System and updating the currently running version of the SDP Operational System.

There is SSH access to the management server running Core Infrastructure Services. All common operational activities are expected to be fully automated, with any manual intervention started via a web interface such as RunDeck. As such, the SSH interface is expected to be used, as a last resort, when direct access to Platform Services is required.

It is worth noting that all the web interfaces are presenting existing interfaces via a HTTPS based interface, i.e. all functionality presented in the web interfaces should be available to automation via an API.

2.1.2.1 Implementations

Candidates: Kibana, OpenStack Horizon, Grafana, RunDeck

Note most of the candidate technology choices for Platform Services have already implemented a Web Interface component for operators. This component is about exposing those in a controlled and integrated way.
From the current prototyping activities, the following web interfaces are expected to allow a good view into the current state of Platform Services, and allow of any required interventions:

- Core Infrastructure Services: If OpenStack is chosen, OpenStack's Horizon dashboard would be exposed so operators can inspect the current state of the infrastructure provisioning. It also allows for the temporary creation or modification of the infrastructure as needed, although it is expected automation would be used to quickly remove such modifications.
- Logging and Metrics: Using something like Kibana would allow operators to search through logs stored in Elasticsearch. While metrics could also be monitored using Kibana, if a monitoring service such as Prometheus is chosen it would be more typical to create views into the metrics using Grafana.
- Operational Dashboard: Shows the overall state of the platform and all the state of each of the Platform Services (TODO: reference state diagram later in document). This view into the SDP would leverage whatever metrics technology is chosen, i.e. it could be implemented as a Grafana Dashboard, rather than a custom in-house implementation.
- Operations as a Service: All operations are expected to use automation. Web interfaces such as Ansible AWX and RunDeck [RD06] help expose those workflows

to operations staff. All the available operationally automation scripts will have gone through quality assurance prior to being available in the Operations as a Service dashboard.

2.1.3 Core Infrastructure Services

Provides an API to manage all the physical hardware resources (i.e. compute, storage and networking hardware), i.e. an API such that Orchestration Services can request the appropriate physical compute resources, along with a running operating system that has correctly configured networking and storage.

There are the following interfaces to Core Infrastructure Services:

- Infrastructure Provisioning API is accessed by both Orchestration Services and the User Interface Services to both provision and deprovision infrastructure resources. Due to a lack of standards in this area, that API will be specific to what technology is down selected.
- Orchestration Services provision and maintain the Core Infrastructure Services running on dedicated hardware.
- The Logging and Metrics Service is used to aggregate logs and metrics from the Core Infrastructure Services.

On first inspection it may appear there are dependency issues here. Firstly Orchestration Services both provision and require the Infrastructure Provisioning API and the Logging and Metrics both run on infrastructure provided by Core Infrastructure Services and monitor Core Infrastructure Services.

It is worth noting that the Logging and Metrics service will be running on infrastructure provided by Core Infrastructure Services. This is not expected to be a problem, as log aggregation of the Core Infrastructure Services does not need to be available until after the Core Infrastructure Services have been provisioned.

The diversity of the hardware that needs to be managed is discussed in detail in the Hardware Decomposition View [RD04]. In particular there are several different networks that a system may or may not need to be connected. Some of the networks use physically separate infrastructure, such as the low latency network for example, while some networks use technologies such as VLANs or overlay networks to keep several streams of traffic separate. There will likely be several different types of physical servers that are specific to one or more server personalities, some of which may have accelerators, or specific storage hardware such as NVMe disks.

When considering the robustness of the SDP Operational System, it is worth noting that the uptime of the hardware provisioned by Core Infrastructure services is largely independent of the uptime of the Infrastructure Provisioning API.

2.1.3.1 Implementations

Implementations: OpenStack, OpenNebula, CloudStack

If we look at OpenStack [RD07] as a potential candidate for Core Infrastructure Services, the following services and APIs are used to provide the functionality required:

- Authentication and Authorization: e.g. Keystone
- Orchestrating compute resources: e.g. Virtualization from Nova
- Orchestrating baremetal servers: e.g. Ironic
- Orchestrating network resources: e.g. Neutron
- Store compute images: e.g. Glance
- Orchestrating remote block storage resources: e.g. Cinder
- Automation to maintain the system, e.g. Kolla-ansible and Kayobe
- Note there are also supporting services (such as database, queues, storage for glance) that the Core Infrastructure Services depend on. While they may be similar to some of the supporting services needed by SDP Services, they will not be shared between SDP and Core Infrastructure Services.

However, we can consider the fact that this API is split into multiple endpoints and the fact they require a set of databases and queues are all implementation details of the chosen technology. The important aspect is that there is an API that allows Orchestration Services to provision the hardware needed to run the software.

2.1.4 Logging and Metrics Services

These services, gather and aggregate logs and health metrics of servers and software components (for both Platform Services and the SDP Operational System), and alerts on any problems that can't be automatically resolved. An example of a problem that should be quickly detected, and operators should be quickly alerted to, is the failure of an Receive and Real Time server. With these tools in place, problems should be spotted early, and understood and resolved quickly. It should be possible to automatically resolve some issues with minimal interruption to the instrument and executing science workflows.

Here is a description of the interfaces highlighted in Figure 1:

- All Platform Services use the Logging and Metrics interface to aggregate their logs and report metrics on the health of each service. This is the same interface that is used by components in the SDP Operational System to aggregate logs and report metrics.
- The User Interface services have an interface to query the logs and metrics that is used by the Dashboards Service that visualise the log and metric data (e.g. Kibana has access to the Elasticsearch API, Grafana would access the Prometheus API)
- Orchestration Services are responsible for provisioning and maintaining the Logging and Metrics Service

There are two main architectural patterns for metric collection, pull vs push. Given both have been proven useful at large scale, the choice of technology used to provide the metrics service will dictate which pattern is adopted.

2.1.4.1 Implementations

Implementations:	OpenStack Monasca, Elasticsearch/Logstash/Kibana, Prometheus/Grafana

For the pull metrics model, let us consider how things would be structured using Prometheus. The Prometheus server is responsible for pulling metrics at regular intervals from various REST endpoints. An agent, called Node Exporter, exposes hosts metrics such as CPU and Memory usage over the REST API expected by the Prometheus server. The API is very simple, so its possible that SDP services could expose a HTTP metrics API for the prometheus server to scrape as an alternative to integrating with a plugin to node exporter. Generally counters are scraped at regular intervals, and the rate of change in those counters are the generated metrics that are shown in graphs.

For the push metrics model, let us consider OpenStack Monasca (using Elasticsearch and its metric beat agents would look very similar). In this model an agent, in this case the Monasca agent, pushes metrics to the Monasca HTTP based API. It is possible for software to directly push metrics to the HTTP based Monasca API.

In both models, the metrics are collected (either directly or via an agent), and stored in a way that makes the metrics queryable via a Web Interface exposed by the User Interface Services.

For logs, all current candidate technologies use a push model where an agent pushes logs to a central aggregation point. If OpenStack Monasca were chosen, an agent pushes the logs to Monasca's HTTP API. If Elasticsearch were chosen, filebeat would be used to push logs from each Operating System to logstash for further processing, which in turn passes them to elasticsearch to be indexed and persisted.

It is worth noting how logging can be integrated using Docker containers in many ways. A service can output all logs to standard out, Docker can aggregate all the logs on the local disk. This can then used as a cache while the logs are pushed centrally via an agent, along with information about what service produced the logs and when they were produced. This approach can be used with all the current candidate technologies.

2.1.5 Remote Storage Provisioning

The Remote Storage Provisioning component is responsible for creating Storage Backends and providing SDP Operational Services with the information needed to mount that as a filesystem, so it can be presented to the appropriate SDP Operational System components. The key use of Remote Storage Provisioning is the Buffer component [RD01]. It is anticipated there will be multiple Storage Backend components, each using different types of physical storage hardware (for example SATA-SSD and/or NVMe) in order to meet reliability and cost constraints.

As illustrated in Figure 1, Remote Storage Provisioning has the following interfaces:

- Storage Provisioning Interface allows SDP Operational System components like the Data Island Controller to create Storage Backends and obtaining the connection information to mount them as a filesystem

- Orchestration Services are responsible for provisioning the Remote Storage Provisioning API and its dependencies.
- Logs and metrics for this services are sent to the Logging and Metrics Service

Please note while this storage may be used for staging Data Products to be passed to Delivery, Platform Services are not responsible for long term persistence of those data products. That responsibility lies with the Long Term Storage component listed in the SDP Operational Services C&C, which is assumed to be an off-the-shelf HSM (Hierarchical storage management) solution.

2.1.5.1 Implementations

Implementations:	Lustre, BeeGFS, CephFS and other Parallel File Systems (PFS). OpenStack Manila (with PFS as back-ends), Ceph + RADOS Gateway, OpenStack Swift

To illustrate how the Remote Storage Provisioning helps provide a File System Interface on top of the infrastructure provisioned by Core Infrastructure Services, we can consider a specific example of using OpenStack Manila with the storage backend of GlusterFS. Orchestration Services would be responsible for creating servers using Core Infrastructure Services that contain hardware appropriated for the targeted storage tier (i.e. performance or capacity focused). It is possible that storage hardware is configured using the Block Storage service that is part of the Core Infrastructure Services. On the newly provisioned servers GlusterFS will be installed and configured. For each physical disk, you create a Gluster brick. You then create a set of Gluster volumes that combine the correct number of bricks for the required capacity and performance. OpenStack Manila configuration is then updated to create a share service that has the GlusterFS cluster configured. When a filesystem is requested, Manila assigns one of the volumes for that requested file system share, allowing the user to retrieve the details needed to mount that GlusterFS volume. Once mounted on the processing node, the processing node has access to the filesystem it needs. This prototyping is discussed in the Prototyping and referenced memos [RD02].

Further prototyping effort is planned to understand the opportunities underlying the use of object storage, exposed to the SDP Operational System via a file system like interface.

2.2. Relations and Their Properties

All relations are shown and described in the Primary Representation.

2.3. Element Interfaces

2.3.1 Orchestration

A lot of the internal interface connections relate to Orchestration. These interfaces are there to represent how automation will be used to maintain both Platform Services and the SDP Operational System, based around Continuous Integration and Continuous Delivery methodologies, aligned with the SAFe approach.

2.3.2 Off-the-shelf APIs

Where possible Platform Services make use of existing software. As no down-selections have yet been made, we cannot specify the exact nature of all the interfaces. Prototyping work has looked at possible options to uncover any architecturally significant variations that may be present. For example, if we downselect on OpenStack the Infrastructure Provisioning API would be the OpenStack Nova API. The lack of well supported standard interfaces for any of these technologies is the key driver behind this approach.

2.3.3 Logging and Metrics

All services have their logs aggregated and metrics collected. What is monitored is expected to evolve over time, as configured by Orchestration Services. Services outside Platform Services are monitored in the same manner as all Platform Services are monitored.

2.3.4 External APIs

Figure 5 details the external APIs and how they are used. Many of these will be off-the-self APIs, such as that detailed in the Compute Provisioning interface.

2.4 Element Behaviour

2.4.1 Bootstrapping of SDP Operational System

To better understand the responsibilities of each component in Platform Services and how it relates to all the external interfaces shown above, we now consider various aspects of starting up the SDP Operational System.

Let us consider the very first time Platform Services and the SDP Operational System are started. First we must power on the management server that hosts the SSH access, part of the User Interface Services. From SSH access, a bootstrap automation script, part of Orchestration Services, can be executed that is able to start the Operations as a Service dashboard. With the Operations as a Service dashboard available, the operator is able to run the automation configured previously by Orchestration Services to bootstrap the rest of the SDP Operational System. Once complete, all Platform Services should be available, and the SDP should be started. For this discussion, SDP has successfully started when we have the minimum viable set of services running, including the SDP Tango interfaces that are used by the Telescope Manager to trigger high-level SDP state changes including processing.

As part of the “bootstrapping” process, Orchestration Services are responsible for starting the Core Infrastructure Services. Once available, the Core Infrastructure Services are used to provision infrastructure as needed for both other Platform Services and the minimal viable services needed to start the SDP Operational System. For example, the infrastructure needed to run the Logging and Metrics service is provisioned, and the required services are deployed, configured and started by the Orchestration Services. The User Interface Services, and managed services (such as Database and Queue) required by SDP services are all started in a similar way to the Logging and Metrics Services.

Part of the Orchestration Services responsibility is maintaining various shared services needed by the SDP Operational Services, such as Database and Queues. The SDP

Operation Services are told how to connect to those services by the appropriate connection information being registered via the Service Discovery interface.

2.4.2 SDP Operational System use of Provisioning Interfaces

The next major consideration for Platform services is the way the Processing Controller component of the SDP Operational Services provision the compute and storage needed to execute the currently scheduled workflows. This is done via the Compute Provisioning and Storage Provisioning interfaces. For more details see the element catalog entry for Execution Control in the SDP Operational System C&C [RD01].

To understand the role of the Remote Storage Provisioning interface and its relationship with the other Platform Services, we again consider how this is bootstrapped by the Orchestration Services. First the appropriate type of hardware for the targeted storage tier is provisioned via the Core Infrastructure Services. On this hardware the appropriate Storage Backend (such as, for example, Lustre, Ceph, GlusterFS, etc) is then configured. This backend is added to the Remote Storage Provisioning service, which is able to expose appropriate connection information via the Storage Provisioning interface, such that the Processing Controller is able to mount the (potentially remote) filesystem and expose it to the appropriate workflows. For more information on the Buffer and how it relates to the Storage Provisioning service see the Buffer element catalog entry in the SDP Operational System C&C [RD01]

2.4.3 Platform Services State

While each service has a very complex set of different states, the high level state of the overall Platform Services can be summarized by the following states:

- **Available**: all expected functionality working as expected, not this does not imply there is enough capacity available to run SDP,
- **Error**: a maintenance action is required by the operator before Platform Services will be available.
- **Unavailable**: Platform Services cannot be contacted to determine its current state

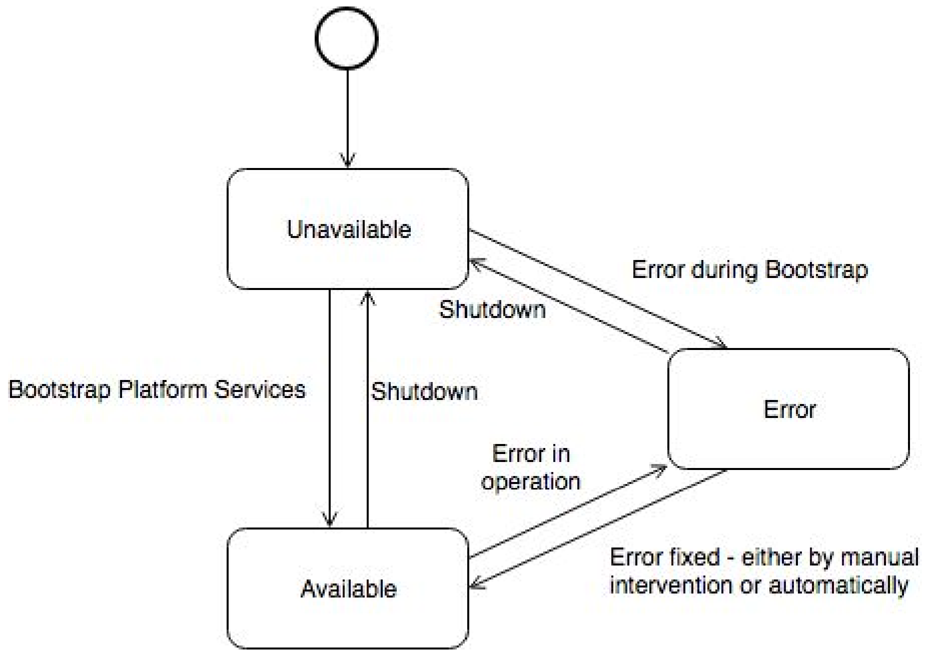
Unavailable
Error during Bootstrap
Shutdown
Bootstrap Platform Services
Shutdown
Error
Error in operation
Available
Error fixed - either by manual intervention or automatically

Figure 2: Platform Services States

2.4.4 Compute and Remote Storage Provisioning Interface State

SDP Operational System (and Operators) need to know the current available resources to both the Storage Provisioning and Compute Provisioning APIs.

For the Storage Provisioning interface, we can consider a unit of storage capacity to map to some logical unit of Buffer capacity, for each Storage Backend tier (n.b. there could be a capacity optimised tier and a performance optimised tier). For Compute Provisioning interface we can consider a unit of compute capacity to be a combination of RAM, CPU and (optionally) an accelerator, that may be specific to a Receive and Real Time Processing or Processing Server personality.

Each capacity unit for Compute Provisioning and Storage Provisioning is in one of the following states:

- **Assigned**: currently provisioned
- **Available**: available to be provisioned
- **Error**: needs operator intervention to make it available
- **Unavailable**: known to be unavailable, and in a low power state (i.e. the compute hardware is likely to be powered off)

The state of the Compute Provisioning and Storage Provisioning can be can be determined by aggregating the number of units of each state.

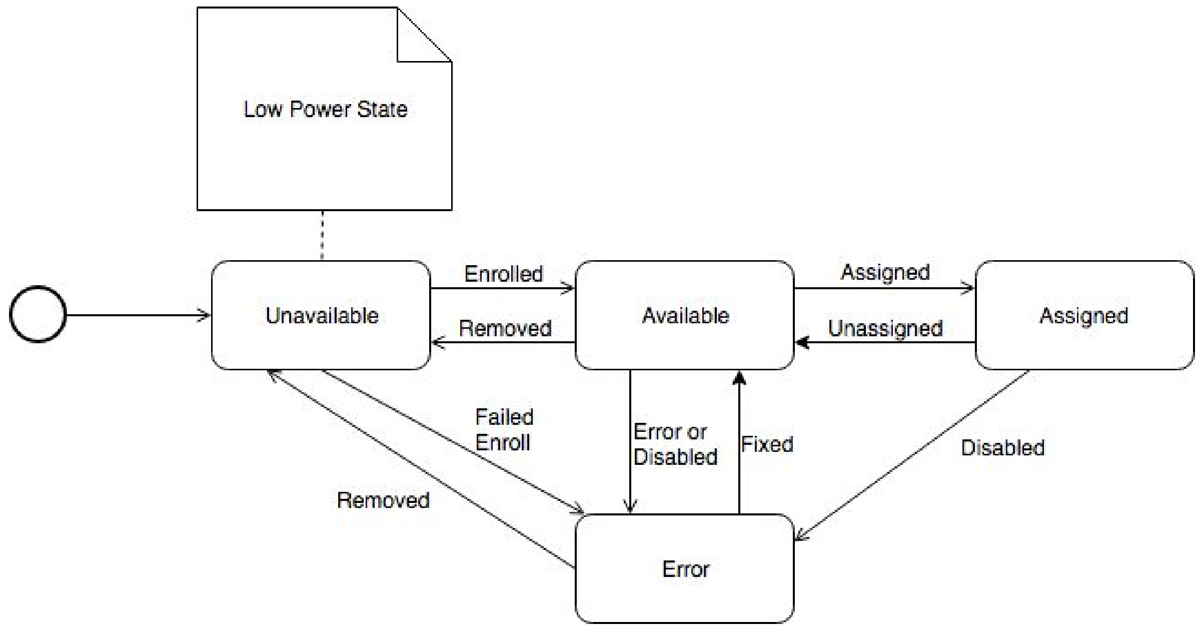

Figure 4: A state machine showing the different states of a Provisioning Unit

Further to Figure 4, here is a description of the transitions that may occur during a typical Provisioning Unit lifecycle:

- A Provisioning unit is enrolled, making it available for Assignment. Underneath, this means one or more servers controlled by the Core Infrastructure Provisioning API will move to the assigned state.
- When a unit is requested, that becomes Assigned to a specific workload.
- If a problem is spotted, a provisioning unit may be disabled, marking it in the Error state.
- If fixed the unit can return to be available state, or it may be removed from pool of provisionable resources
- Once no longer required, an Assign unit can return to the Available state.
- Should a low power mode be required, available provisioning units are removed from the system, i.e. the underlying Core Infrastructure Services server will move to the the available state, meaning it is in a low power mode.

We do not currently distinguish between assigned and pre-staged resources. For example, it may be possible to create buffers and push out container images to compute resources while not consuming any RAM. Further prototyping is needed to determine if that distinction will be needed.

2.4.5 Infrastructure Provisioning API Server State

We can consider each of the servers under the control of Core Infrastructure Services to be in one of the following states, focusing on the operator view of what Core Infrastructure Services is doing:

- **Allocated**: server is being used, i.e. has a provisioned instance
- **Available**: server can be considered when provisioning a new instance. Note the server will be shut off, but with its out of band management interface still available.
- **Unavailable**: server cannot be considered when provisioning a new instance, due to an expected/known reason

- **Error**: server cannot be considered when provisioning a new instance, and needs operator intervention before returning to either an Allocated or Available state.

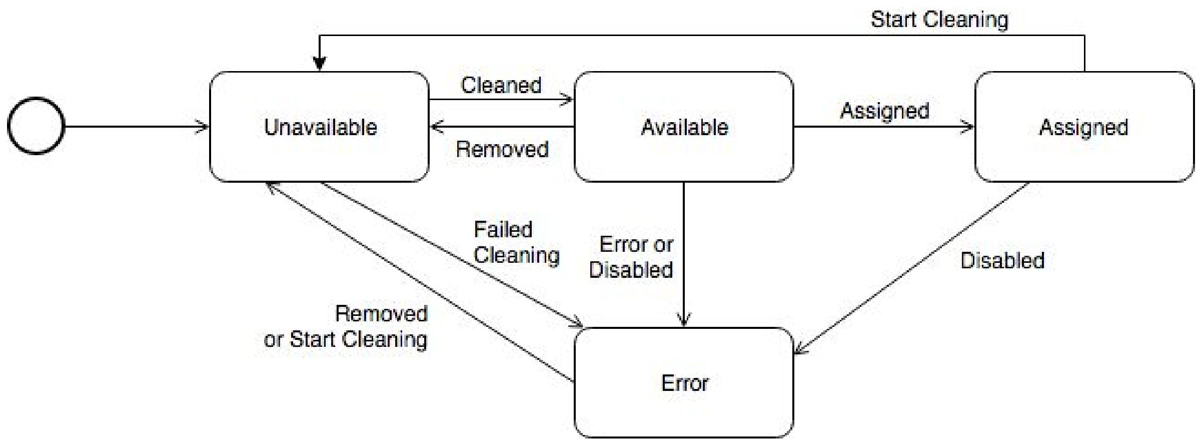

Figure 3: Infrastructure Provisioning API Server State

It is worth noting that whether the server is powered on or off is independent of the above states, except when in the available state where the server might be put into a a reduced power state. If we look at OpenStack Ironic, the server will be powered off, but the server's out of band management interface would still be available to power the server on when required. So after successfully bootstrapping the SDP, all servers managed by Core Infrastructure Services would be in the Available state, sitting powered off, but ready to be powered on when needed.

User Interface Services are expected to provide a dashboard to visualise the current state of all the infrastructure (such as a pie chart showing what state all the infrastructure servers are currently in). Note this is for information only, it does not help communicate the state of SDP Operational System. For example all servers could be allocated but powered down, which would mean no SDP services are currently running.

3. Context Diagram

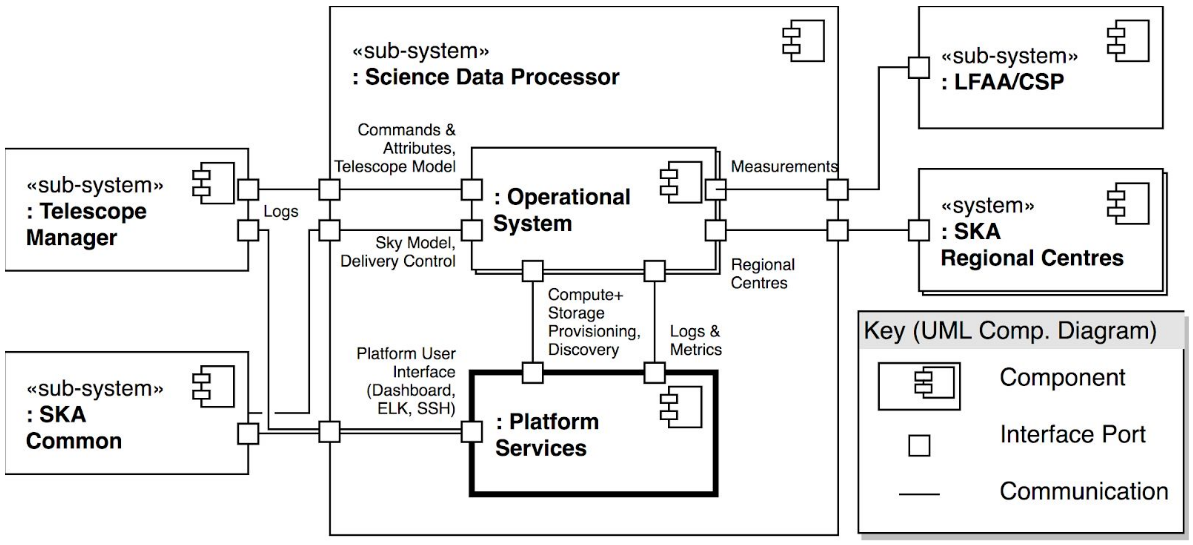

Figure 5: Platform Services Context Diagram. For more details on how the Platform and Operational System interact via the Compute Provisioning and Storage Provisioning interfaces, see Figure 2: Execution Control Decomposition Diagram in the System C&C [RD01].

4. Variability Guide

We now discuss the ways in which the architecture supports particular variations.

4.1 Flexibility in mapping Software to Hardware

The core of what Platform Services allows is flexibility in how software is deployed onto the physical hardware resources. As noted in the hardware decomposition view [RD04] there will be range of server personalities and over time various generations of hardware spread across refresh cycles. These details can be largely abstracted away from the SDP Operational System via the Compute Provisioning, Storage provisioning and Discovery interfaces provided by Platform Services.

Ideally we would adopt standard interfaces for Core Infrastructure Provisioning, but non exist. However, most APIs follow similar patterns, such that adopting the Software Defined Datacenter approach, it is possible to port between the different technologies that are chosen. Orchestration must take care to separate the provisioning of infrastructure from the configuration of the running operating system provided by the Infrastructure Provisioning API. The orchestration to configure the SDP Operational System should be portable between all candidate Core Infrastructure Services. This is expected to be increasingly significant when we consider Regional Science Centres.

4.2 Cost control and re-use

Platform Services is architected to help the SDP Operational System reuse existing software, where possible via protocols and standard interfaces. A wide variability of component solutions is anticipated to be available.

4.3 Shared Services exploit existing software to aid later harmonisation

Roughly following the dependency inversion principle, the SDP Operational System does not attempt to manage details such as running databases, rather the platform details with deploying and maintaining the databases. The dependency between the platform and the software is purely the interface used to talk to database, with service discovery acting as a kind of Dependency Injection. One big advantage of this approach is that as common patterns emerge across the SKA, harmonisation can occur, so the SKA can be operated more efficiently.

It is worth nothing this approach is not restricted to databases, Metrics and Logging are treated in a similar way, as are Queue services. It is expected there will be many different types of supporting software that can be managed in this way. Where it exists, existing SKA wide harmonisation, orchestration tooling and operational patterns should be adopted.

It may be possible services such as databases are provided by a service offered by the same vendor providing the Core Infrastructure Services, for example public cloud platforms frequently offer database as a service offerings. Architecturally, that can be considered an implementation detail of how Orchestration Services provisions the database. It is Platform Services that is responsible for the provisioning and maintenance, it is not the responsibility of the SDP Operational Services. However, depending on these tight integrations, this does increase the work that is needed to operate the SDP Operational System across multiple types of Core Infrastructure Services.

4.4 Authentication and Authorization

Each component is responsible for integration with the SKA wide AAAI System. It is expected that standards based interfaces such as SAML or OpenID Connect will be used to integrate Platform Services with the external AAAI system.

Future work is needed to better define this relationship, including any required interfaces to Figure 1. It is possible some of Platform Service's interfaces will not require integration with the external AAAI system.

4.5 Provisioning and Orchestration Options

There are two levels of provisioning specified in the architecture. First the Infrastructure Provisioning API (the lower level, used by Orchestration Services) and second the Compute and Storage Provisioning APIs (higher level, used by services outside Platform Services, such as the SDP Operating System's Execution Control component.)

Currently prototyping efforts are exploring the use of OpenStack Ironic baremetal provisioning for the Infrastructure Provisioning API, OpenStack Manila for the Storage Provisioning API and Docker Swarm for the Compute Provisioning API. However the architecture does not dictate the level of abstraction for each of these APIs, rather it specifies which users needs each API must meet. Further prototyping will need to determine the best fit Orchestration and Provisioning strategies, such as virtualization, baremetal and containers. It is very likely a mix of strategies will be needed to meet all the varied performance and utilization needs.

Because there are no widely adopted standard APIs for these Provisioning components, the exact API for each of these interfaces is not fully described, because it is expected to dictated by the technologies chosen to implement each interface.

For Metrics, there are two key approaches in the various available technologies: pull (e.g. Prometheus) vs push (e.g. OpenStack Monasca). For Log collection, all current candidates map to a push based model.

4.5 Storage Backends

The main architectural restriction on the Storage Provisioning Interface is the need to provide a file system like abstraction (in particular accessing stored files via a hierarchical namespace). In addition there will be a separate namespace for each data islands, and each namespace will map to one of multiple tiers of storage. This leaves the implementation free to choose the best storage backend to match the needs of a particular storage tier.

It is not expected that the requirement on a filesystem like interface will significantly restrict the choice of storage technology. For example, if an object storage system were to be chosen, that can still have files references via a hierarchical namespace. Most of the files are written two by a single host and have multiple remote (i.e. not on the same host as the writer) readers.

5. Rationale

The key drivers of the architecture are described in detail in the SDP Architecture Overview [RD05]. In respect of Platform Services, a model used extensively in Cloud is considered. In particular this addresses:

Cost Control and Re-use: Although the SDP presents a new software architecture there is a significant requirement not to "reinvent the wheel" wherever possible. This is highly desirable for cost control but also schedule control and reliability of reusing, where possible, tested software modules. The SDP architecture has therefore been designed with this in mind - for example the architecture of the Platform Services closely follows open-source products such as OpenStack while not making a downselection to a particular technology so that an appropriate solution can be adopted at the time of construction. This should make SDP easier to support and operate, and enables the use of a wide range of standard tools and processes to develop and maintain the SDP Operational System that executes on the platform.
Availability: The SDP is required to have high levels of availability and resilience. This has led to the designing of the SDP around, wherever possible, loosely coupled components with stateless control with supporting services encapsulated in the Platform.
Modifiability: The adoption of a Cloud model, provides extensive CICD techniques allied to the separation Execution Control and Platform Services. This supports the enablement of modified and new workflows to be developed in an operational context by observatory staff.
Performance: Unlike the most common deployment of Cloud, typically using virtualisation and network isolation, the emergence of baremetal provisioning and host networking will ameliorate any performance penalties associated with such technologies, while maintaining to a large extent the benefits of Cloud models.

Another driver for Platform Services is a look towards further SKA level harmonisation, where it is likely we will want a centralized operations team that have responsibility for the maintenance of services like Kafka that can be shared between multiple components, and services like databases where a consistent set of operating patterns around backup and recovery from HA failures would be an advantage. The current catalog of services in Platform Services is the current set of best candidates to consider for this treatment. A key point here, is that these services being part of Platform Services, do not need to be provisioned and maintained by SDP components such as Execution Control [RD01]. It is reasonable to expect the list of services offered will increase over time as we establish more services that are available.

The requirement SDP_REQ_813 considers the availability of SDP Software outside of the SPC context, there is no specific deliberation of this requirement here, however, let's take a moment to consider running SDP Operational Services in a Regional Science Center. It is possible the environment will be very different to the Telescope local compute resources, and will dictate a very different set of technology choices for Platform Services. For example, if a public cloud such as AWS were used, it would dictate a different approach to how the platform services are constructed. However, it should be possible to keep all the external interfaces with SDP the same in both contexts, by re-implementing platform services for each specific execution context. However, no prototype work has yet been done to validate this suggested approach.

The requirements together with architectural drivers mentioned above, have led to an architectural representation of the SDP couched in a Cloud-like framework. Services required by the Execution Frameworks, for example, are clearly separated making for an extensible that promotes adoption of standard practise in cloud and clearly defines distinct areas for development for both domain and non-domain functionality, and most importantly , re-use where appropriate. We are currently tracking developments in cloud software stacks, such as OpenStack, as well as prototyping these services with the Performance Prototype Platform. [RD02].

SDP_REQ-818
Software failures of the SDP (TBC-084) software that requires rebooting in order to recover from the failure, shall have a MTTR (recover time) of less than or equal to 10 minutes.

Should a server need to be rebooted, the use of a software defined infrastructure and automated orchestration helps to ensure that the SDP system can be restored in a controlled and efficient way. More prototyping is required to confirm the likely timelines of this process.

SDP_REQ-27
The SDP shall be able to switch between previously scheduled observations within 30 seconds (TBC-061- based on performance allocations of SKA-2133. This is 30s overall and SDP's allocation of this time will depend on feedback from the System Review (Project Office)).

The use of orchestration will allow the pre-staging of workflows and buffers potentially assisting speed up when switching between workflows as needed. Current prototyping of baremetal provisioning has proven too slow for workload switching. However, prototyping has shown it should be possible to use Orchestration automation to quickly switch between workloads running on a server that has already been provisioned. Further prototyping efforts are expected to explore the use of Container Orchestration frameworks to assist this requirement.

SDP_REQ-822
The following design requirements were identified to allow quick recovery from node failures. Detection of node failure and setup of spare node shall be done within TBD seconds. The time will vary according to the amount of data loss during the recovery window and this depends on what the node is doing at time of the failure, i.e. short time for real-time processing and longer for offline processing. Re-routing of real-time (incl ingest from CSP) data streams to a spare node when an ingest node fails to ingest shall be done within TBD

seconds. Changing or updating execution graph following a node failure shall be done within TBD seconds/minutes.

The Logging and Metrics service will be designed so server failure can be easily detected, and can trigger alerts. Orchestration can help ensure the timely and efficient resolution steps to recover from failures in any part of the SDP system.

In a similar way to SDP_REQ-818, the use of a software defined datacenter approach to provisioning will help recover from any server failure.

SDP_REQ-813
SKA would like portability of SDP to mixed software environments

The SDP Operational System software is more portable because it does not depend on specific vendor's hardware variations, instead depending on an OS image Core Infrastructure Services and a standard set of APIs.

For more details on requirements see [AD01]

6. Related Views
No directly related views.

7. References

7.1. Applicable Documents
The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, **the applicable documents** shall take precedence.

[AD01]	SKA-TEL-SDP-0000033_RSP_02E_SDPL2Requirements
[AD02]	
[AD03]	

7.2. Reference Documents

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

[RD01]	SKA-TEL-SDP-0000013 SDP Operational System C&C View, rev 05
[RD02]	SKA-TEL-SDP-0000054 SDP Prototyping Report , rev 05
[RD03]	SKA-TEL-SDP-0000013 System-level Module Decomposition and Dependency View, rev 05

[RD04] SKA-TEL-SDP-0000013 Hardware Decomposition View, rev 05

[RD05] SKA-TEL-SDP-0000013 SDP Architecture Overview View

[RD06] https://www.rundeck.com/open-source

[RD07] https://www.openstack.org/software/

[RD08]

[RD09]

8. Version History

Version	**Date of Issue**	**Prepared by**	**Comments**
05	2018-04-25	J. Garbutt et al.	Prepared for M20 Pre-CDR delivery