SKA SDP: Cloud Native

Kubernetes just ate your PaaS!

February, 2019

piers@catalyst.net.nz

Overview

- Cloud Native
- Kubernetes
- Science as a Service
- The SKA
- Where's your PaaS?

Cloud Native

Cloud Native

Is the embodiment of modern software delivery practices

Supported by tools, frameworks, processes and platform interfaces.

These capabilities are the next evolution of Cloud Computing, raising the level of abstraction for all actors against the architecture from the hardware unit to the application component.

Cloud Native advantages

Cloud Native exploits the advantages of the Cloud Computing delivery model:

- PaaS layered on top of IaaS
- CI/CD – fully automated build,test,deploy
- Modern DevOps – auto-scaling, monitoring feedback loop
- Software abstraction from platform compute, network, storage
- Portability across Cloud Services providers

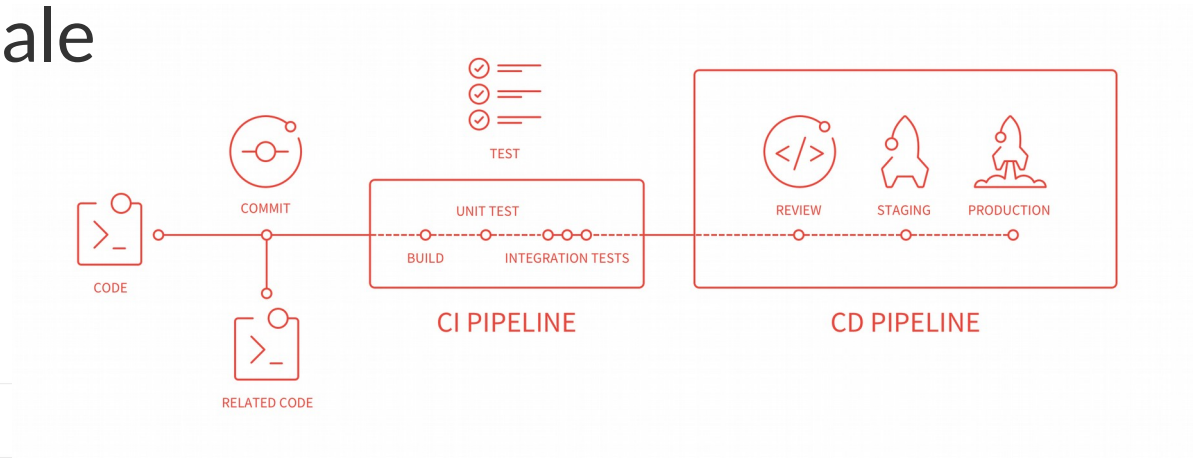
Cloud Native Software Delivery Life Cycle

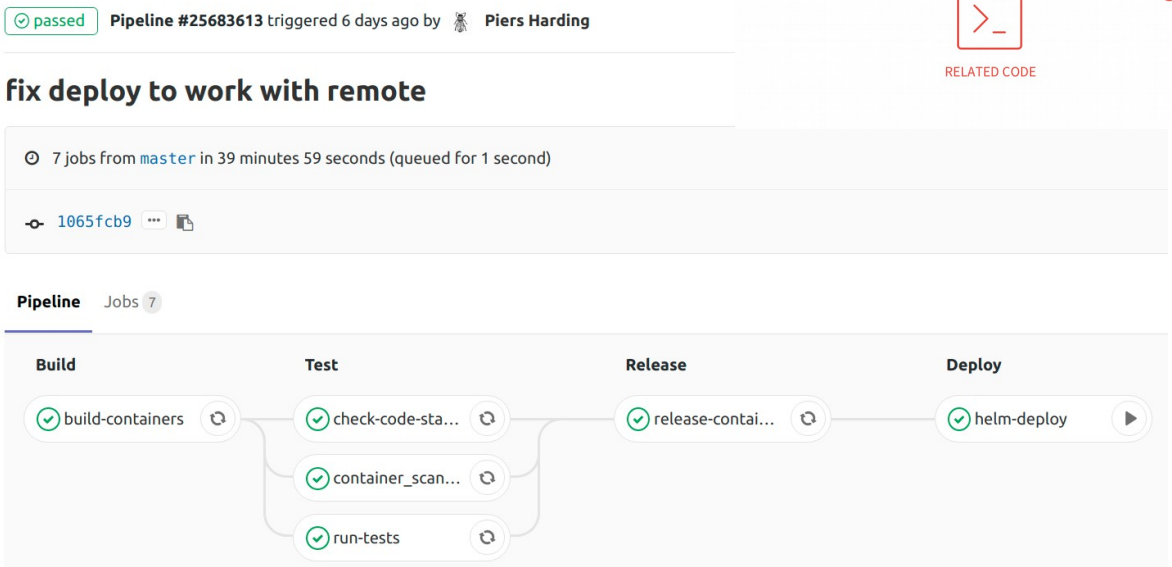
Why Cloud Native SDLC? - cohesion for distributed project

- Codify standards - testing gates
- Code quality, consistency and assurance – CI/CD
- Automation – build AND rebuild (zero day)
- Portability of SDI as well as code
- Reference implementation – best practices, and exemplars
- Engagement – an open and collaborate system “Social Coding Platform”
- Integration with SRC, and other projects – the future platform

SDLC: software life-cycle management

Cloud Native opportunities for automation: Build, test, deploy, scale





Science as a Service – why Cloud Native fits

Delivering the SDP - SaaS

SDP is compelling Science as a Service use case with requirements like:

- 24x7 data capture and processing
- Limited range of processing pipelines
- Storage limitations
- In manufacturing terms - if visibilities are the primary product, then SDP data products are the secondary product that scientists create tertiary products from
- A distributed problem: Software Development and Service Delivery

Cloud Native - SaaS

To realise Science as a Service:

- need a common set of standards for software development and deployment that will scale from the laptop to the super computer
- giving certain guarantees about:
  - Re-usability
  - Portability

This will enable processing to move seemlessly(ish) between facilities, limited only by bandwidth, storage, and processing capacity.

Kubernetes – a unifying abstraction layer

Kubernetes – provides standards

Impact on the actors:

- developers: a known and dependable working environment
- devops: a mechanism for currating and validating compliance of delivered artefacts, and delivery of a heterogeneous software environment
- testing and acceptance: a demonstrable benchmark for good practice and compliance - automation
- security: tracking of core software dependency compliance, with an automated way to upgrade, test and rollout security patching without involving "everyone"

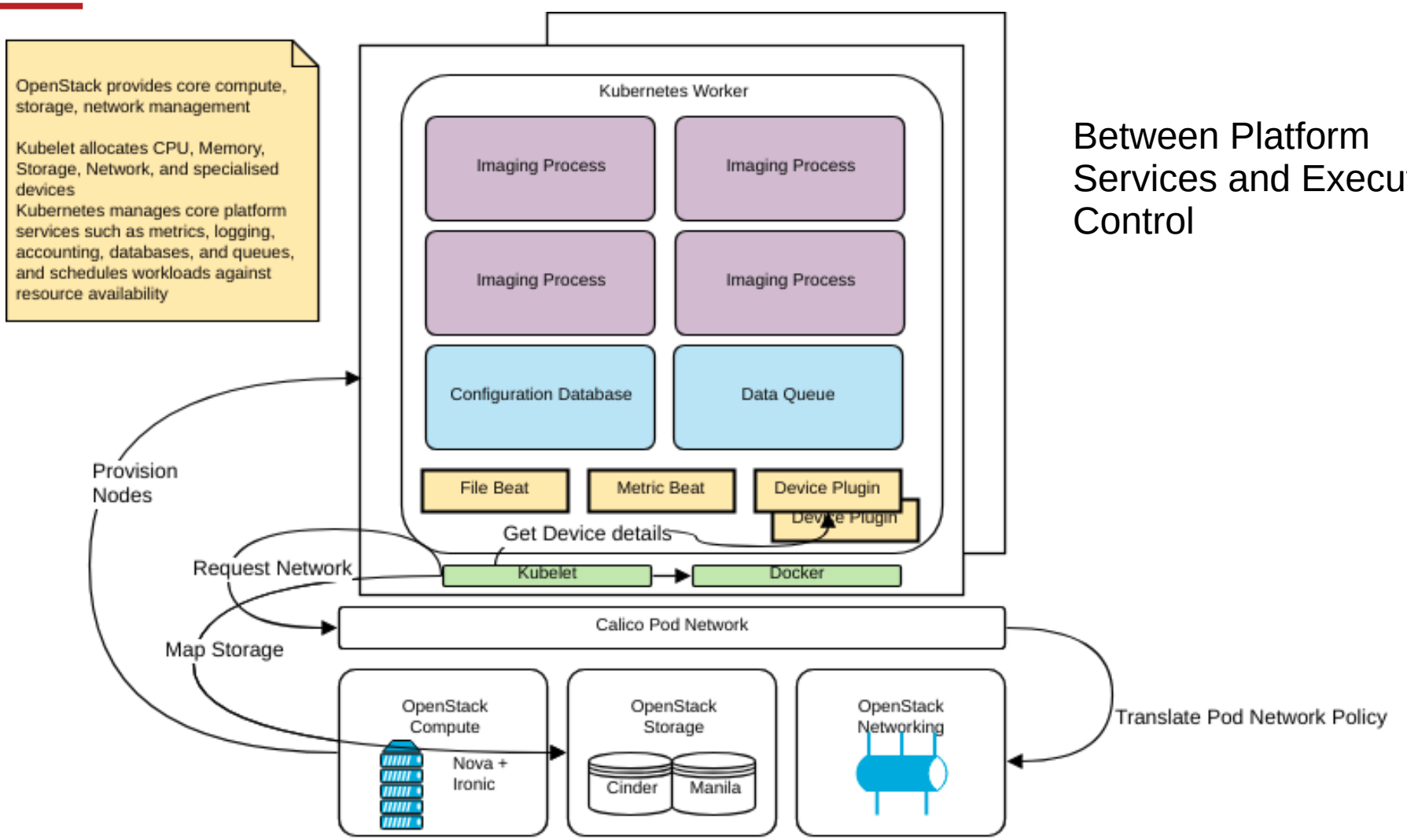
Kubernetes is Cloud Native

Kubernetes is the platform that all the other tooling sits on or integrates with:

- CSI – storage interfaces
- CNI – network interfaces
- Containerd – run time
- Prometheus - monitoring
- Helm - deployment
- Linkerd – diagnostics
- Envoy – service proxy
- Harbor – container registry

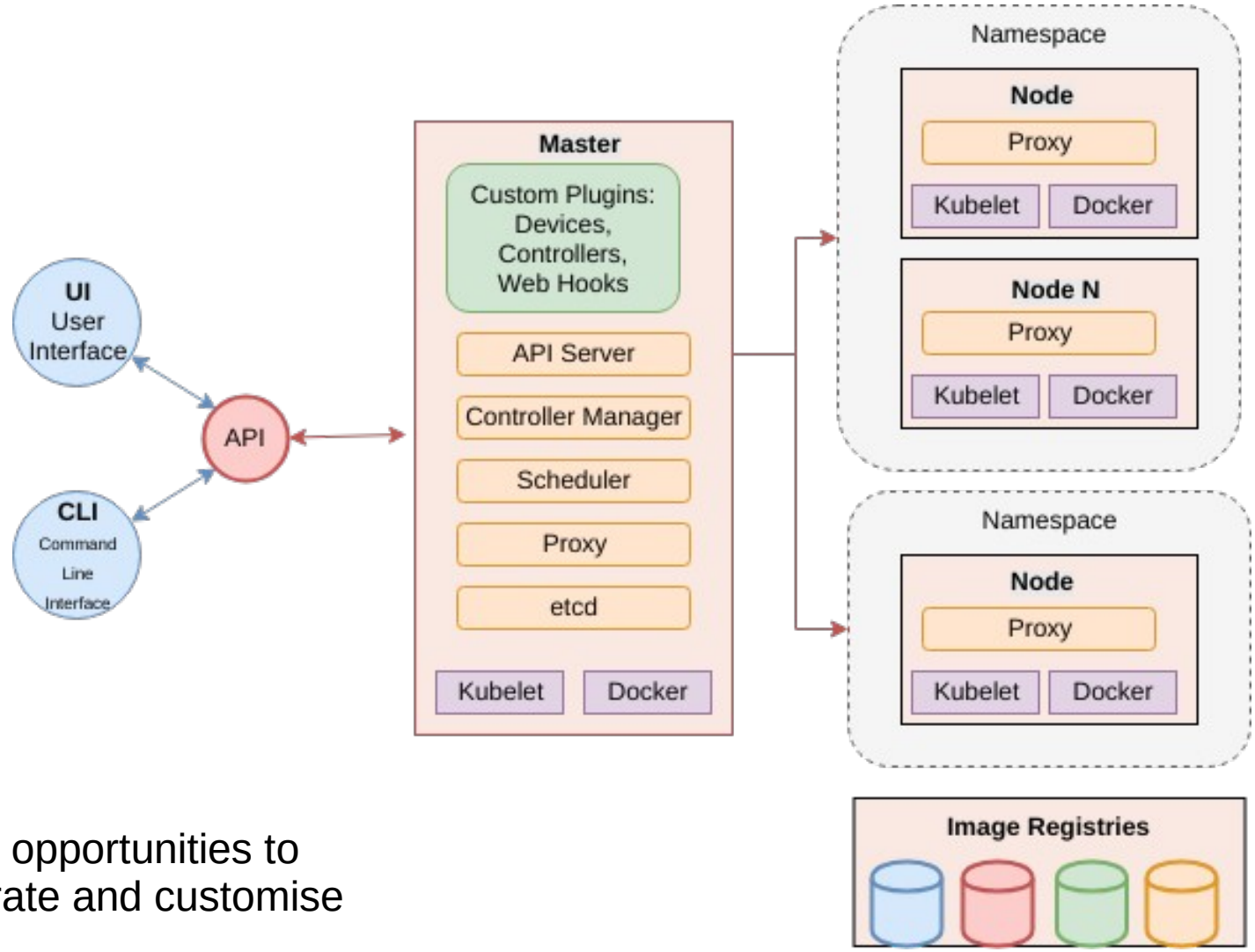
And many more

K8s: Where does it sit in the platform landscape?



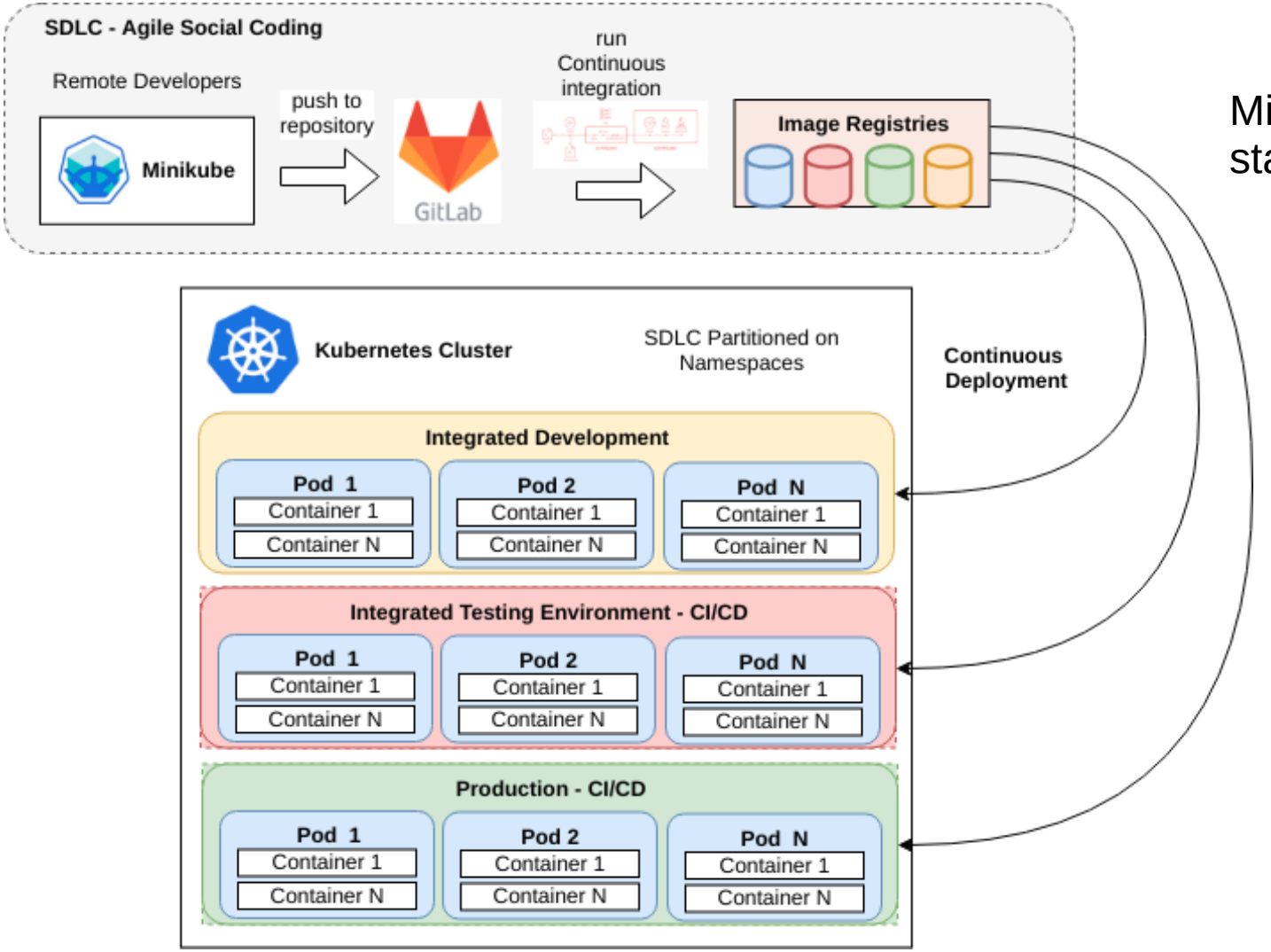
Between Platform Services and Execution Control

K8s: Architecture logical view



Many opportunities to integrate and customise

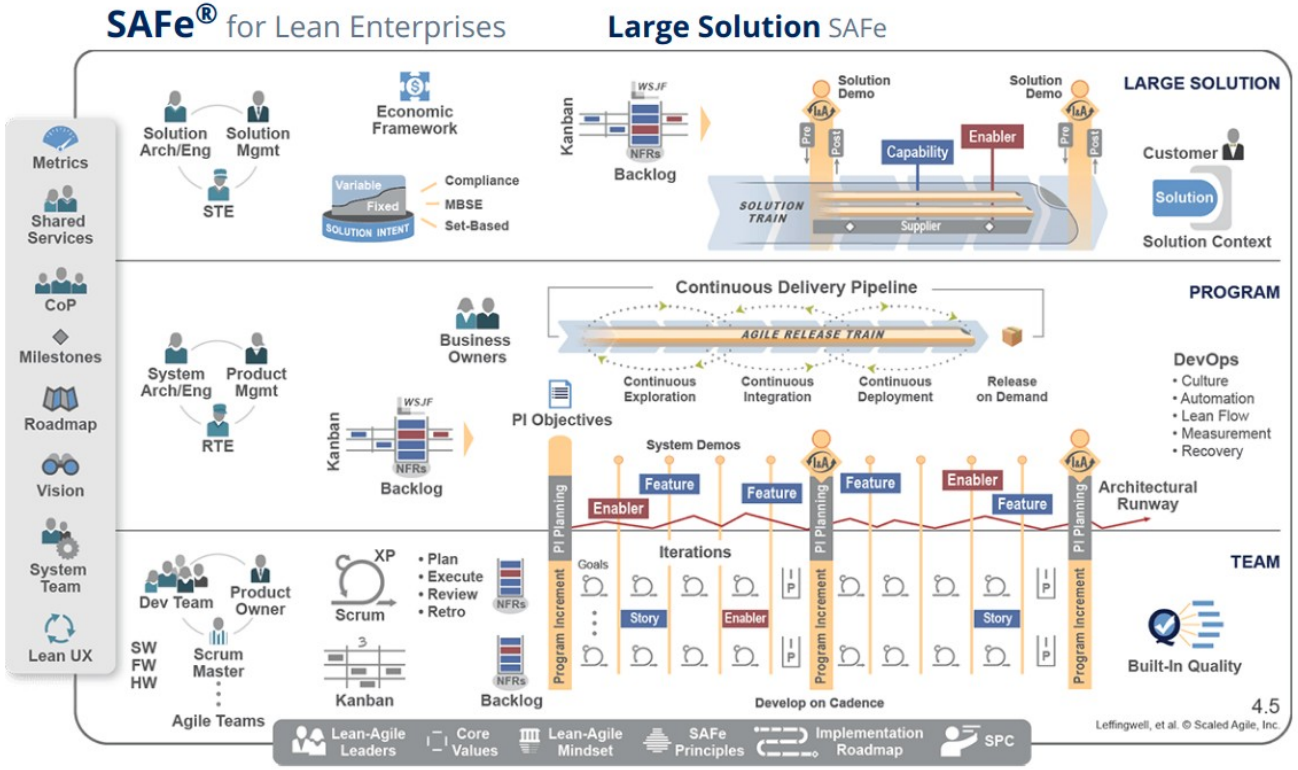
SDLC: software life-cycle management



Minikube is the starting point

SDLC: software life-cycle management

A process and technology that supports SAFe



PaaS - Engagement

Engagement is solved by portability – supplied by k8s

Portability problem:

- Devices
- Storage
- Compute
- Network
- Service primitives

Hardware considerations abstracted from the application

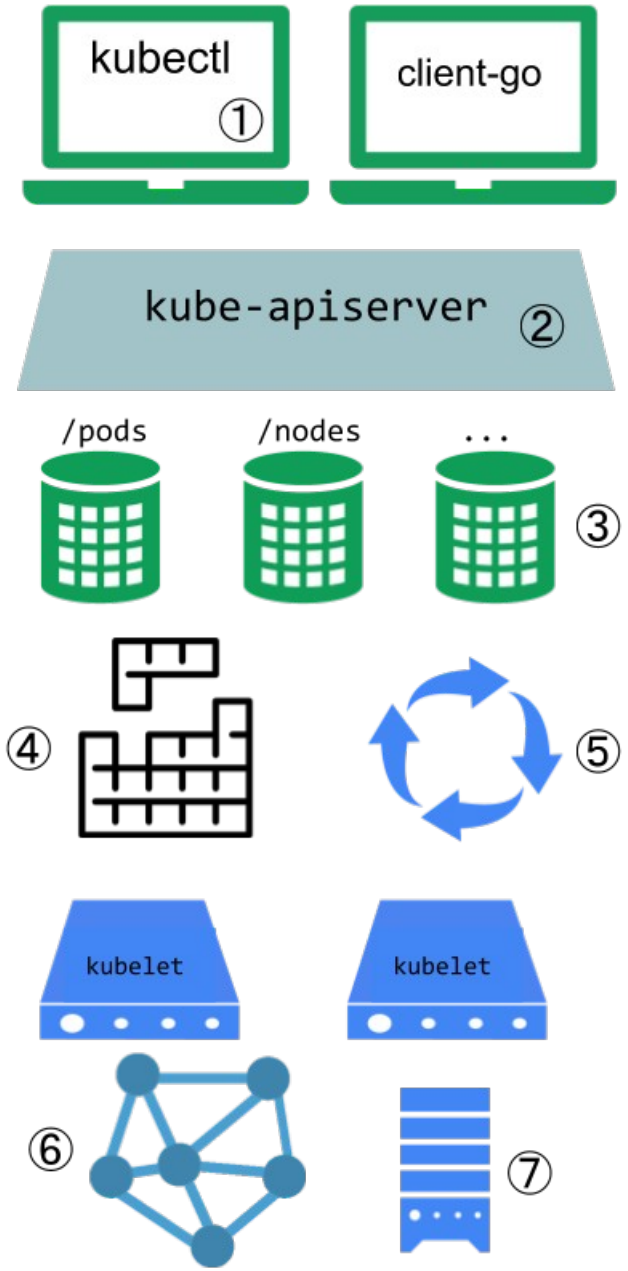
Kubernetes – not just an Orchestration Engine – it's an API

Understanding:

- Deployment options
- Integration options
- PaaS

K8s: Integration Options

1) Kubectl plugins, official client libraries - Keystone
2) API Server extension – ACL, edit requests - Keystone
3) Custom Resources Definitions – partner with (5)
4) Custom schedulers - rare
5) Custom Controllers – API aggregation, pick up custom resources - KubeDB
6) Network extensions – Calico, Kuryr
7) Storage plugins – Cinder storage class, and operator



Kubernetes: Resource Primitives

Applications can be deployed with:

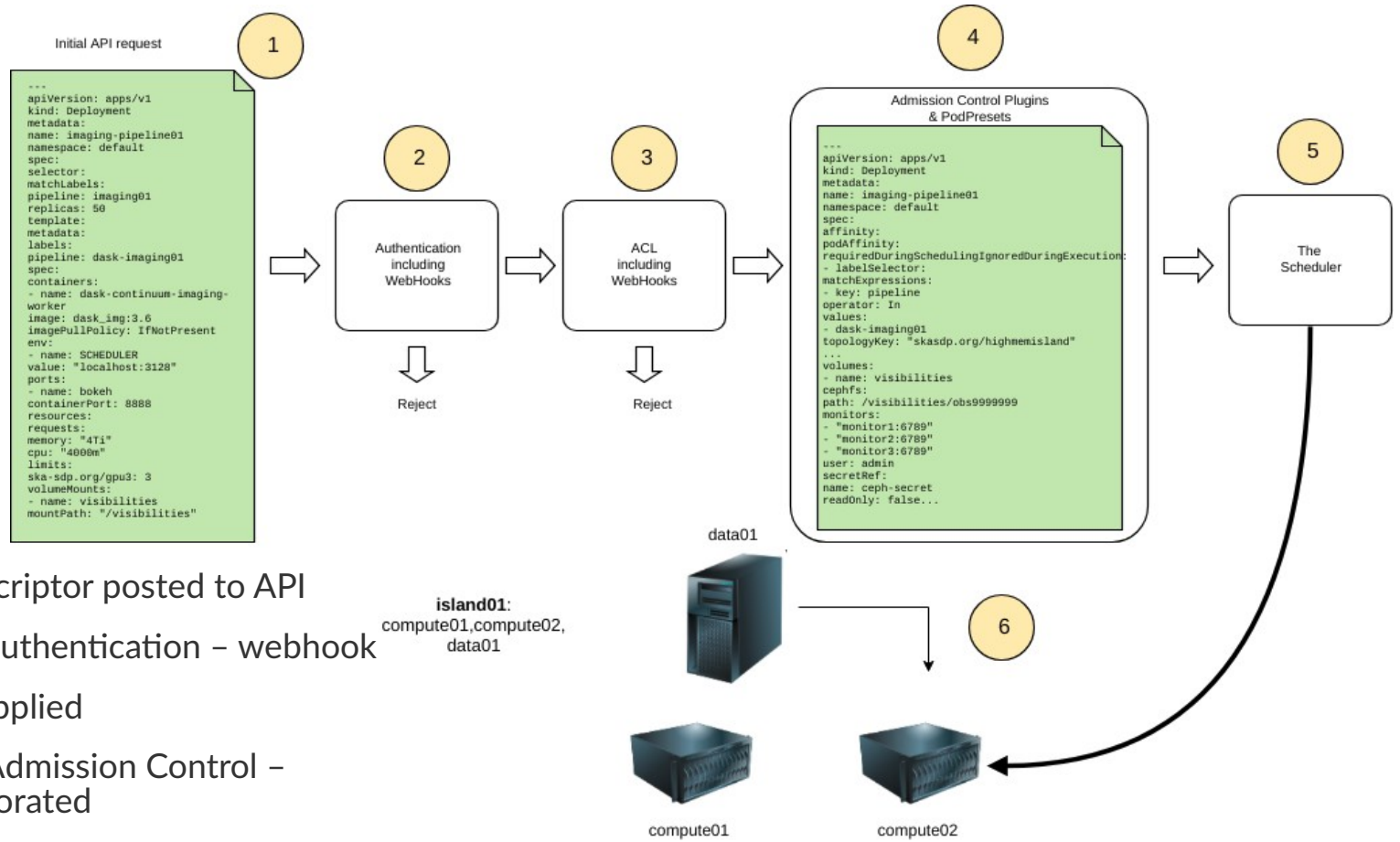
- Application – Pod, Deployment
- Sets – ReplicaSets, StatefulSets, DaemonSets
- Schedule – Job, CronJob
- Network – Service, Endpoint, Ingress, NetworkPolicy
- Storage – StorageClass, PersistentVolumes

Kubernetes: Deployment Options

Applications can be deployed with:

- kubectl run ... - adhoc container execution
- Resource descriptors – application definition and automation as code (including PodPreset templates)
- API Clients – write applications that manage or extend k8s
- Helm – template the templates (charts), with a templating language and configuration control
- Others (meta-tools): Ansible, Teraform, Draft(Azure), Skaffold...
-

Scheduling



1) Deployment descriptor posted to API
2) Passes through authentication – webhook
3) Access control applied
4) Passes through Admission Control – reformatted/decorated
5) Scheduler calculates placement – affinity/anit-affinity
6) Passed to Kubelet which assembles resources and boots container

Operators -
Eg: KubeDB

Custom Resource Definitions

+ Operator

- KubeDB
- MPIJob
- Rook – ObjectStore (Ceph)

Where's your PaaS?

Heroku
Engine Yard
Acquia
AWS
Bitnami
Cloud Foundary
Digital Ocean
...

Replaced by EKS, AKS, GCP, OpenShift ...

What does this mean?

- Platform Services are now abstract services – DB, Storage, Vault, ElasticSearch, Prometheus ...
- SAP & CERN have turned it inside out with managing OpenStack Platform Services from Kubernetes
- Metacontroller – rewrite objects and workflows with scripting
- Kubernetes is eating the stack from Platform Services, up to frameworks
- a DC/OS – resource management and scheduling
- a unifying abstraction layer
- MVP Execution Framework