Getting started with Dynatrace for Kubernetes and OpenShift

Software intelligence for the enterprise cloud

Steve Caron
Solution Engineer
Dynatrace

Matt Reider
Product Manager
Dynatrace

Jeff Neau
Sales Engineer
Dynatrace

dynatrace
Perform

powered by aws

Agenda

1. Kubernetes concepts and architecture
   - Architecture
   - Concepts
   - Distributions
2. Monitoring k8s/ocp with Dynatrace
   - Deploy
   - Entity model & features
   - K8s in context
   - Service and Process Group naming rules
3. Management Zones
   - Organize entities
   - RBAC
4. K8s cluster health monitoring
5. Alerts
6. Additional topics
   - Container monitoring rules
   - Multi-mode monitoring
   - What's coming
   - Q & A

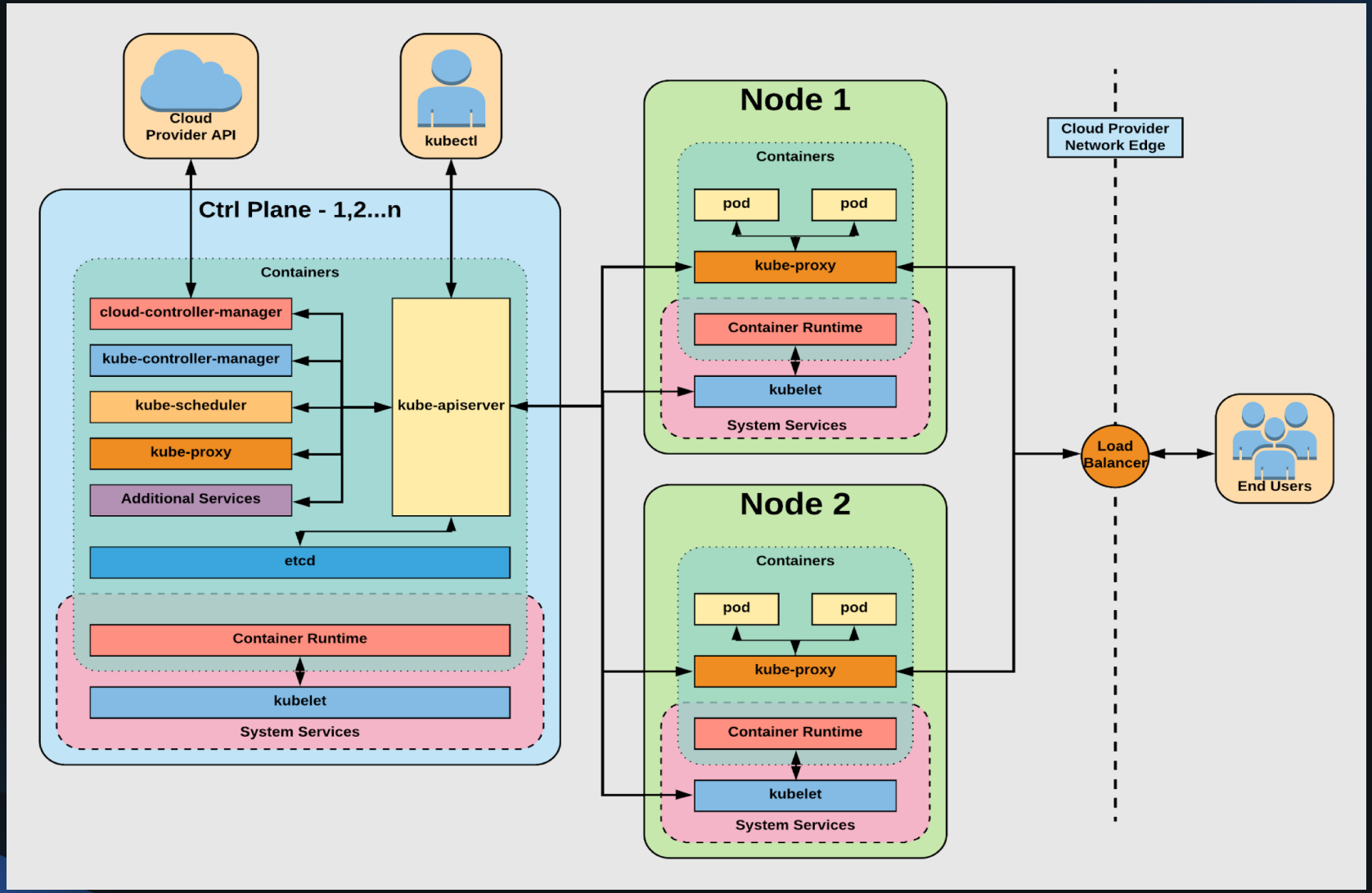
Show of hands : https://pollev.com/stevecaron983

Kubernetes Concepts and Architecture

Major distributions

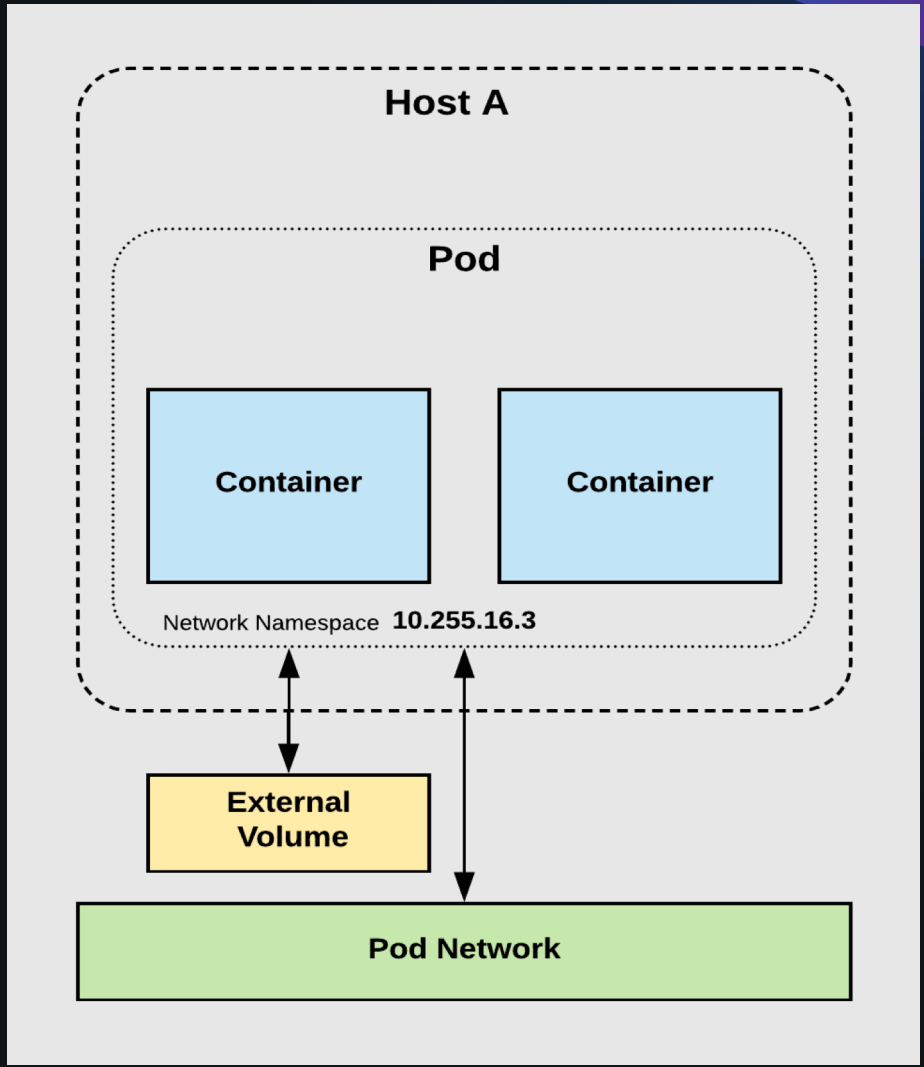
- Canonical Kubernetes
- Docker Kubernetes Services
- Heptio (now owned by VMWare)
- Pivotal Container Services (PKS)
- Rancher 2.0
- SUSE CaaS Platform
- Mesosphere Kubernetes Engine
- Red Hat OpenShift Container Platform
- IBM Kubernetes Services (IKS)
- AWS Elastic Container Services for Kubernetes (EKS)
- Azure Kubernetes Services (AKS)
- Google Kubernetes Engine (GKE)

Architecture overview



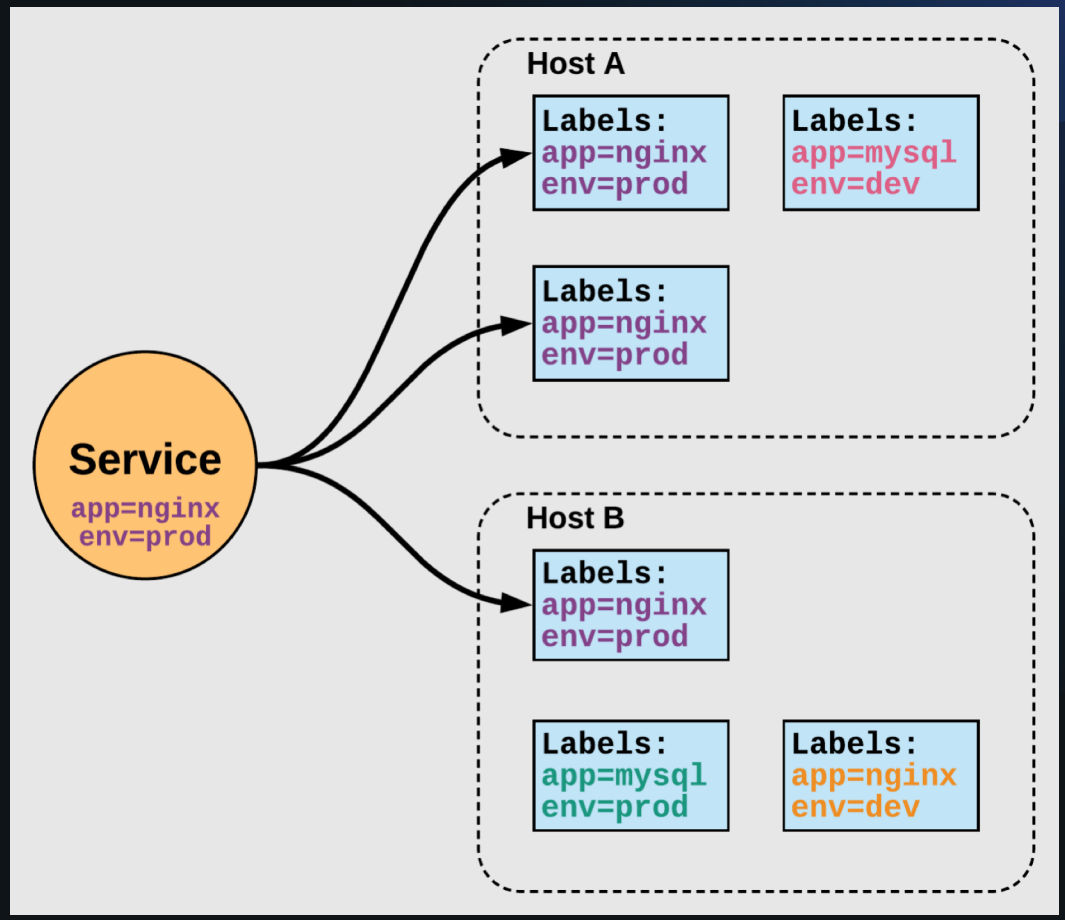
Key Concepts - Pods

- Smallest “unit of work” in Kubernetes.
- Pods are one or more containers that share volumes and namespace.
- They are also ephemeral



Key Concepts - Services

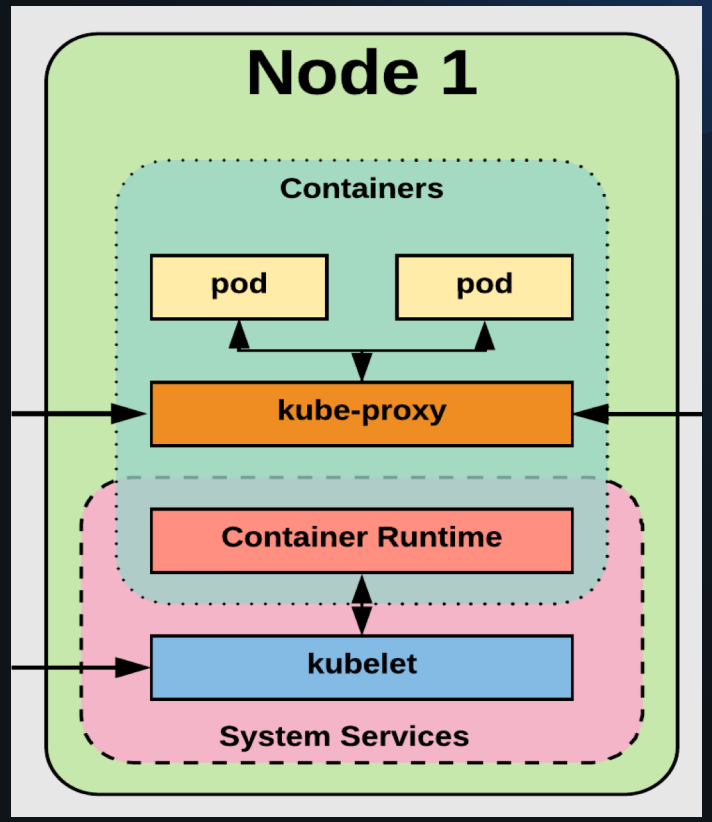
- Unified method of accessing the exposed workloads of pods.
- Think of it as an internal load balancer to your pods.
- How it is implemented depends on the cloud provider or the on-prem config
- Not ephemeral



<service name> <namespace>.svc.cluster.local

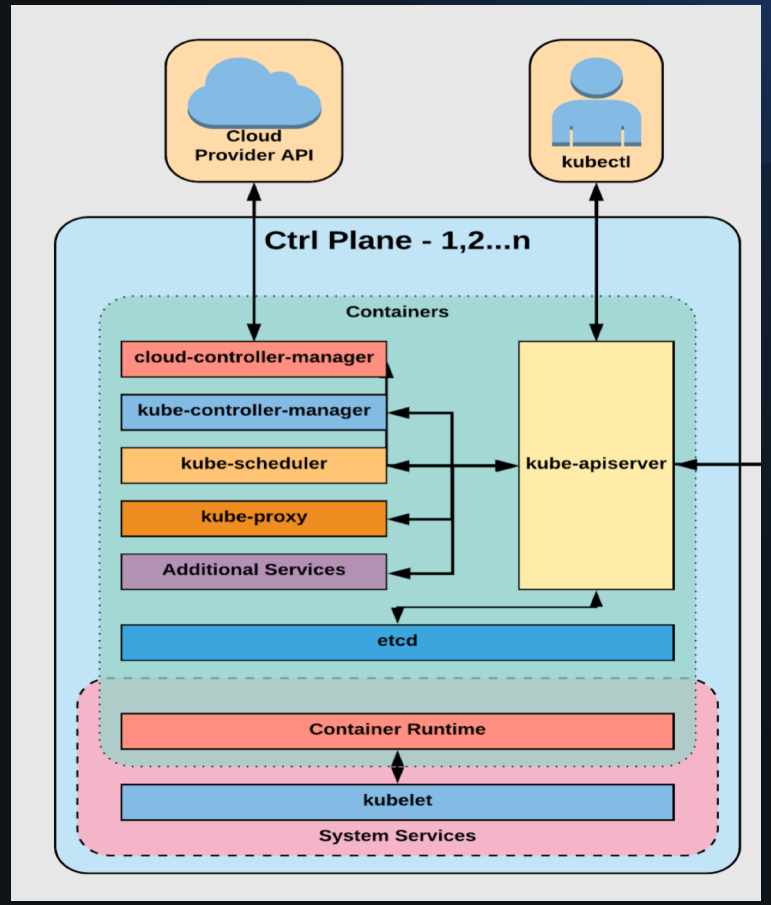
Key Concepts - Nodes

- Worker machine in Kubernetes
- VM or physical machine
- Managed by the control plane (master)
- Node components:
  - kubelet
  - kube-proxy
  - Container Runtime Engine



Key Concepts – Control plane

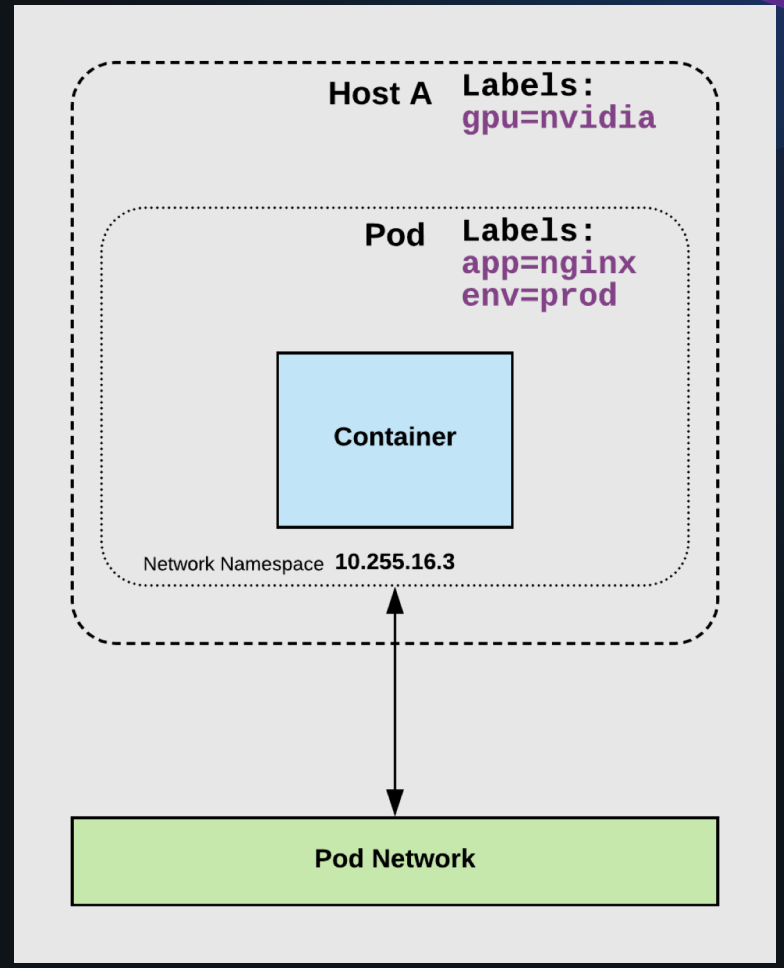
- Components:
  - kube-apiserver
  - etcd
  - kube-controller-manager
  - kube-scheduler
  - cloud-controller-manager
- kube-apiserver:
  - provides forward facing REST interface into k8s itself
  - everything, ALL components interact with each other through the api-server
  - handles authn, authz, request validation



Key Concepts – Labels

- Key-value pairs used to identify, describe and group together related sets of objects or resources.

- Not characteristic of uniqueness

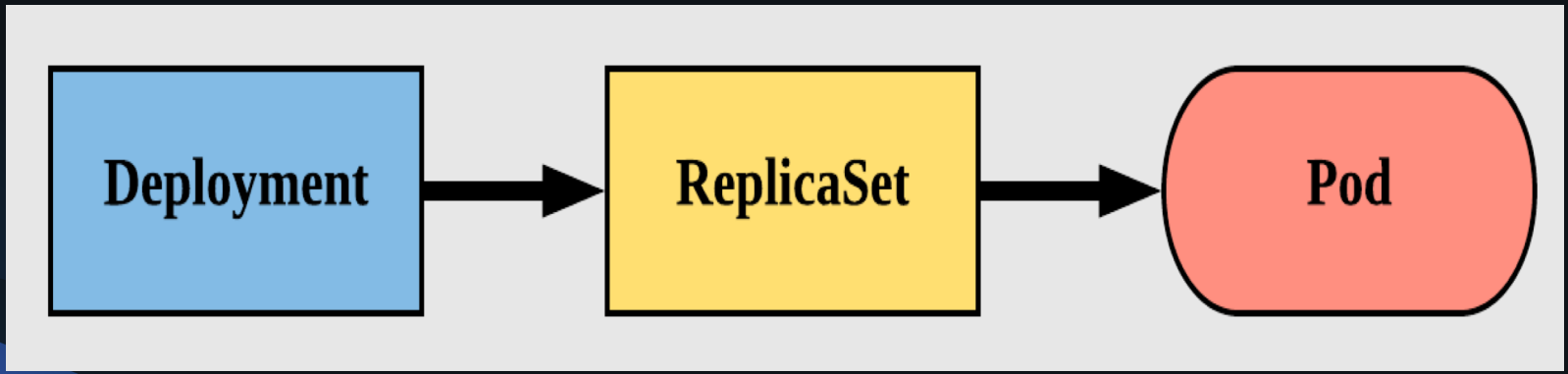


Key Concepts – Replicasets

- Primary method of managing pod replicas and their lifecycle
- Includes their scheduling, scaling, and deletion
- Their job is simple: Always ensure the desired number of pods are running

Key Concepts – Deployments

- Way of managing Pods via ReplicaSets
- Provide rollback functionality and update control
- Each iteration creates a unique label that is assigned to both the ReplicaSet and subsequent Pods.



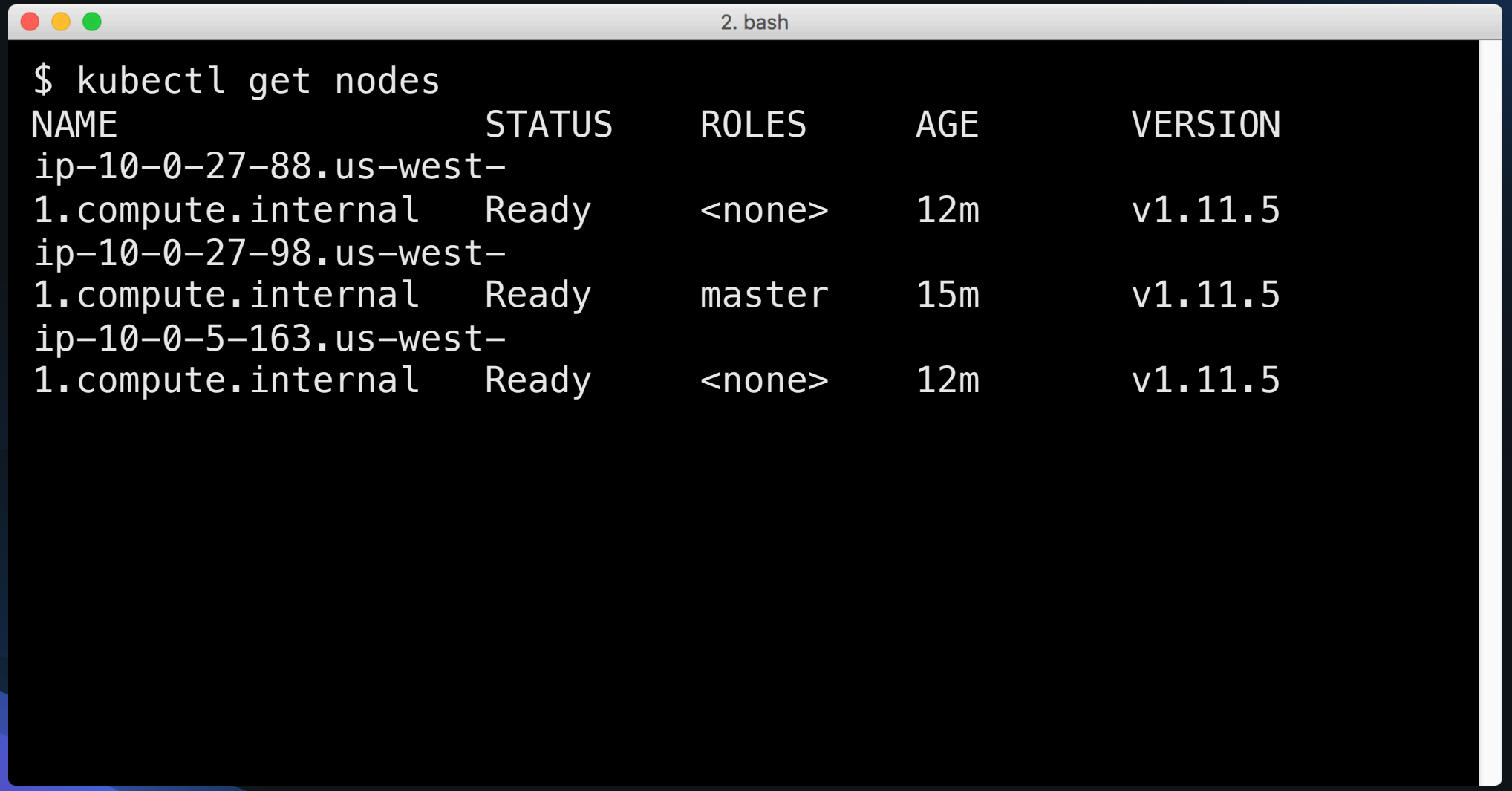
Key Concepts – Daemonsets

- Ensure that all nodes matching certain criteria will run an instance of the supplied Pod
- Are ideal for cluster wide services such as log forwarding or monitoring

Key Concepts – Secrets

- Decouple configuration from application
- Ideal for username/passwords/tokens, certificates or other sensitive information that should not be stored in a container
- Encrypted at rest within etcd
- Stored on each worker node in tmpfs directory

Key Concepts – kubectl (CLI)



Kubernetes Concepts Cheat sheet

Nodes
- are instances of underlying OS (typically Linux)
- virtual or physical machine (bare-metal k8s)
- host one to many *Pods*

Pods
- are (co-located) groups of *containers*
- a Pod's containers run *on the same host (node)*

Labels
- assign *identifying metadata* to *objects* (Pods)
- are used to organize, group or select objects

Annotations
- assign *non-identifying metadata* to objects
- are used for arbitrary data (build, debug info)

Deployments/Replicasets
- select a Pod object using *labels*
- ensure a given number of *Pod replicas*
- handle *scale-up/down* and *rolling updates*

Services
- select a Pod object using *labels*
- abstract a (replicated) set of Pods by *IP, port(s)*
- provide *domain name* and simple *load balancing*

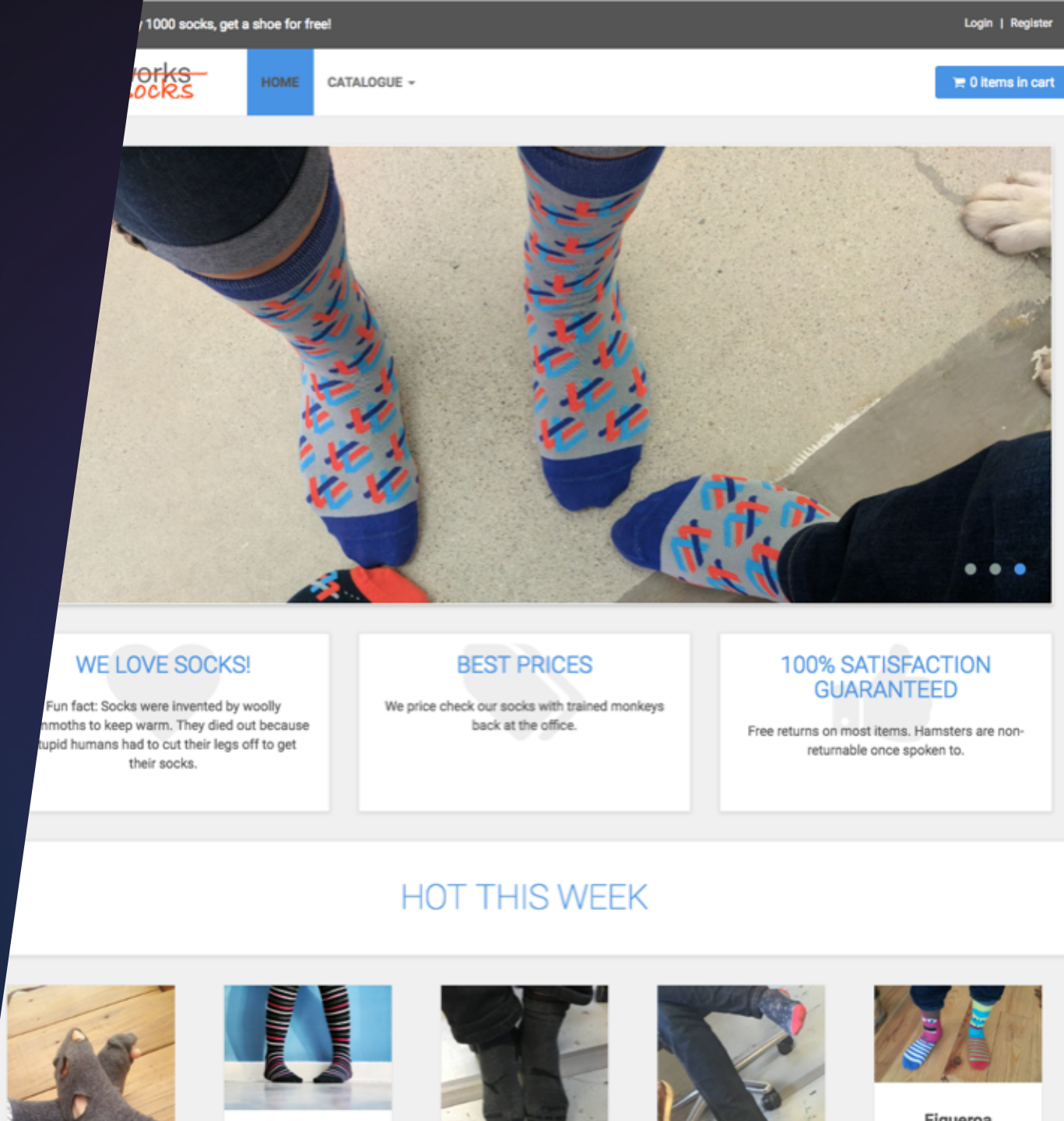
Ingress
- expose services to the (cluster) outside world

Kubernetes concepts also available from the CLI:
$ kubectl api-resources

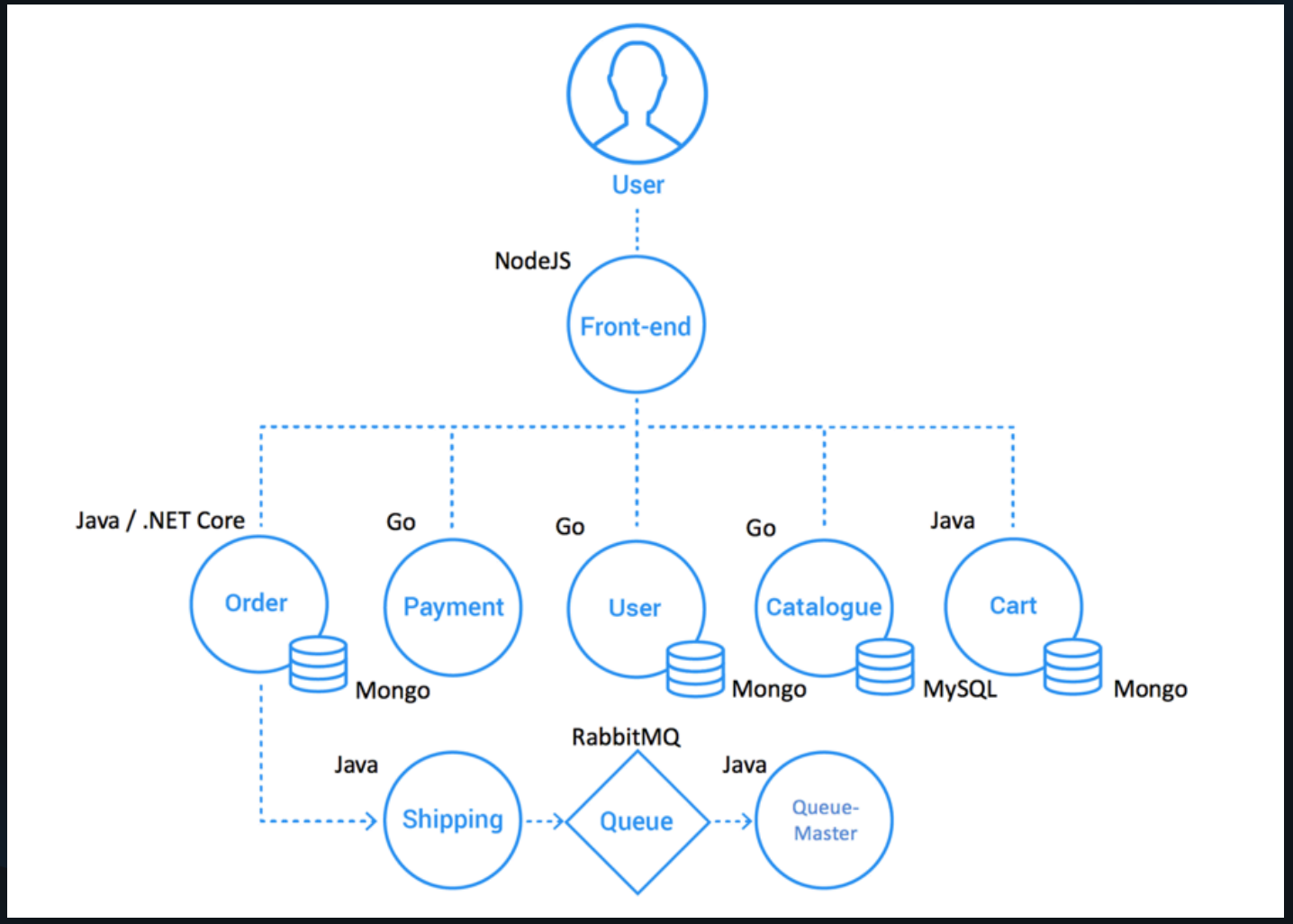
The Sock Shop sample app

Sock Shop Application

- Based on : http://socks.weave.works
- Open source microservice sample app
- Polyglot
  - Java Spring Boot
  - Go kit
  - Node.js
  - RabbitMQ (order queuing)
  - Mongo, MySQL (data stores)



Sock Shop architecture



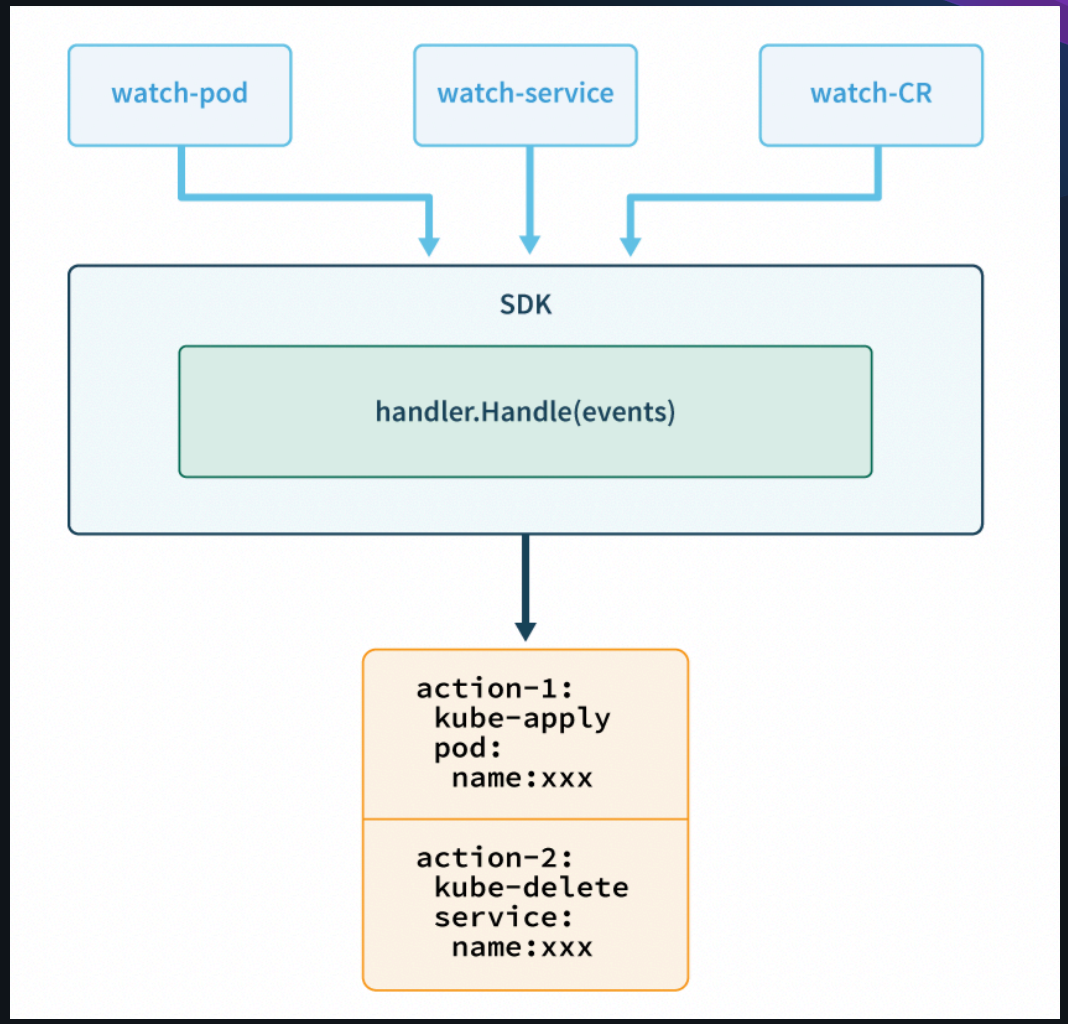
Exercise #1
Deploy the Sock Shop app

Monitoring Kubernetes with the OneAgent Operator

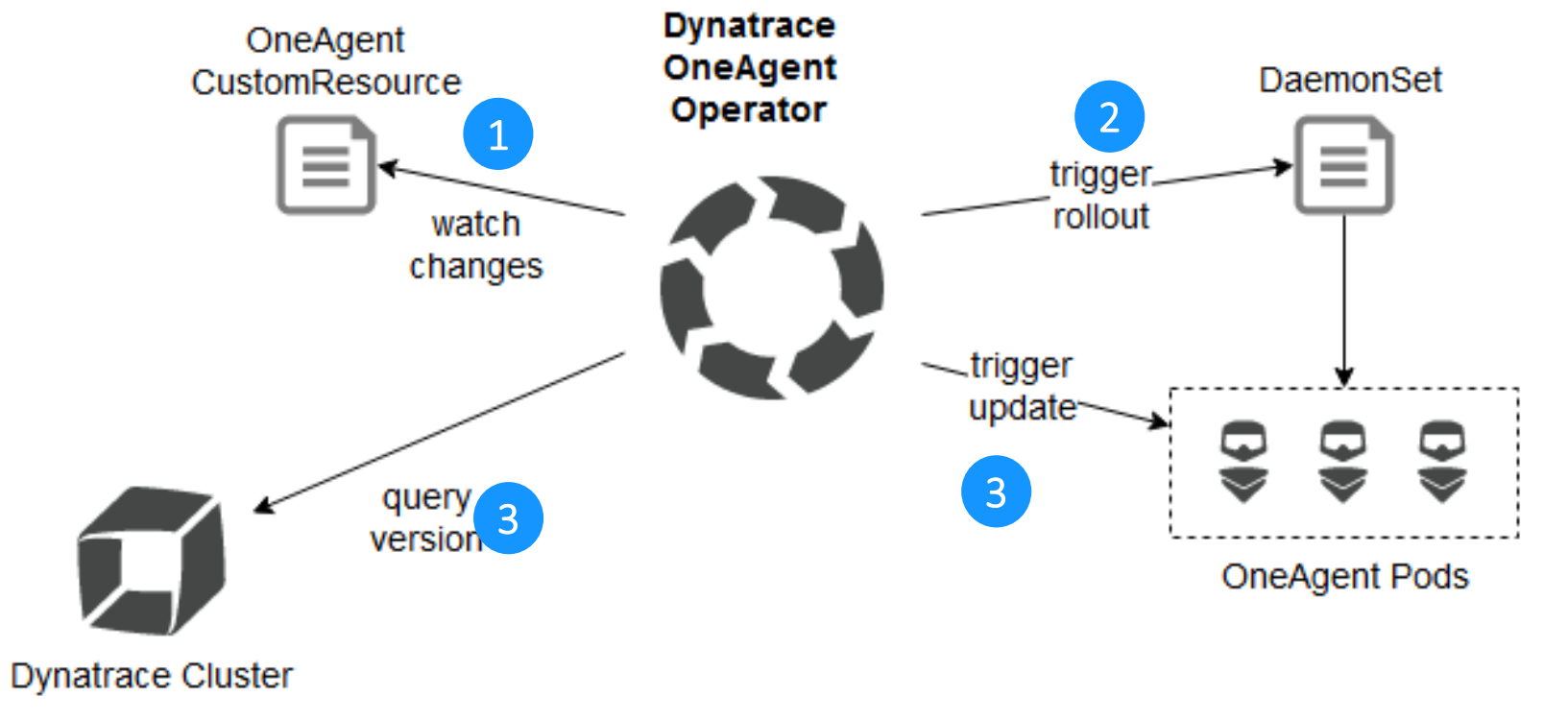
What is an Operator?

OPERATOR FRAMEWORK

- Open source toolkit to manage Kubernetes native applications in an effective, automated, and scalable way.
- *"Human operational knowledge in software"*
- Reliably manage application lifecycles as a first-class kube native object



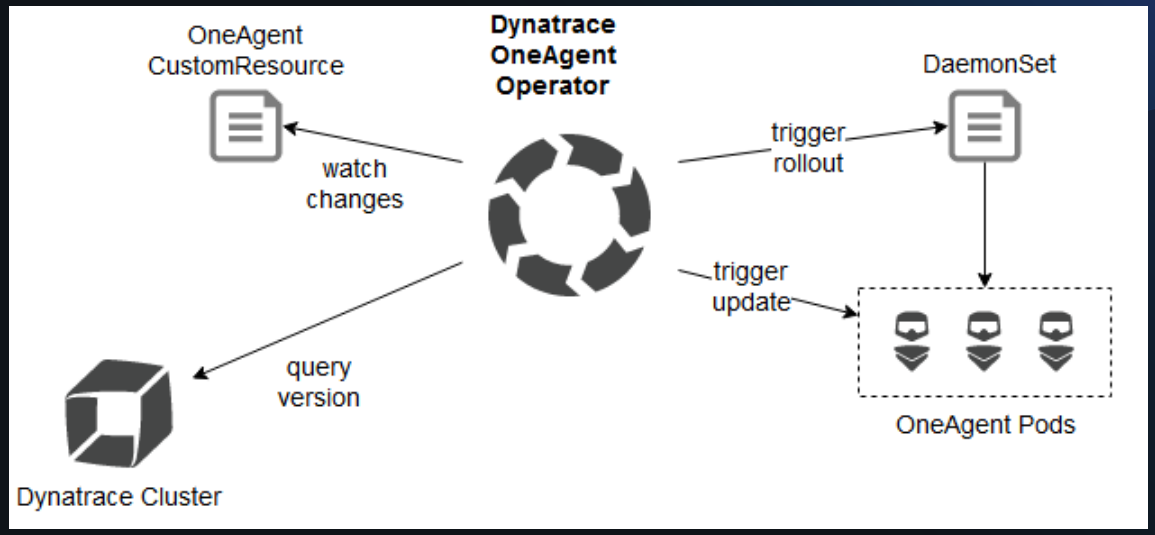
OneAgent Operator in action



1. Watches for custom resources of type OneAgent
2. Takes care of OneAgent deployment via DaemonSet
3. Updates OneAgent to the latest version available

OneAgent Operator value add

- Manage Dynatrace OneAgent deployments
  - no access to host needed
  - fine-grained control of OneAgent roll-out
  - supports tolerations so you can deploy on tainted nodes.
- Up-to-date monitoring
  - OneAgent updates are performed automatically, as soon as they're available.
  - Operator takes care of recycling all pods that have not yet picked up the latest version.
- Validated : available on Operator Hub
  - https://operatorhub.io/operator/dynatrace-monitoring



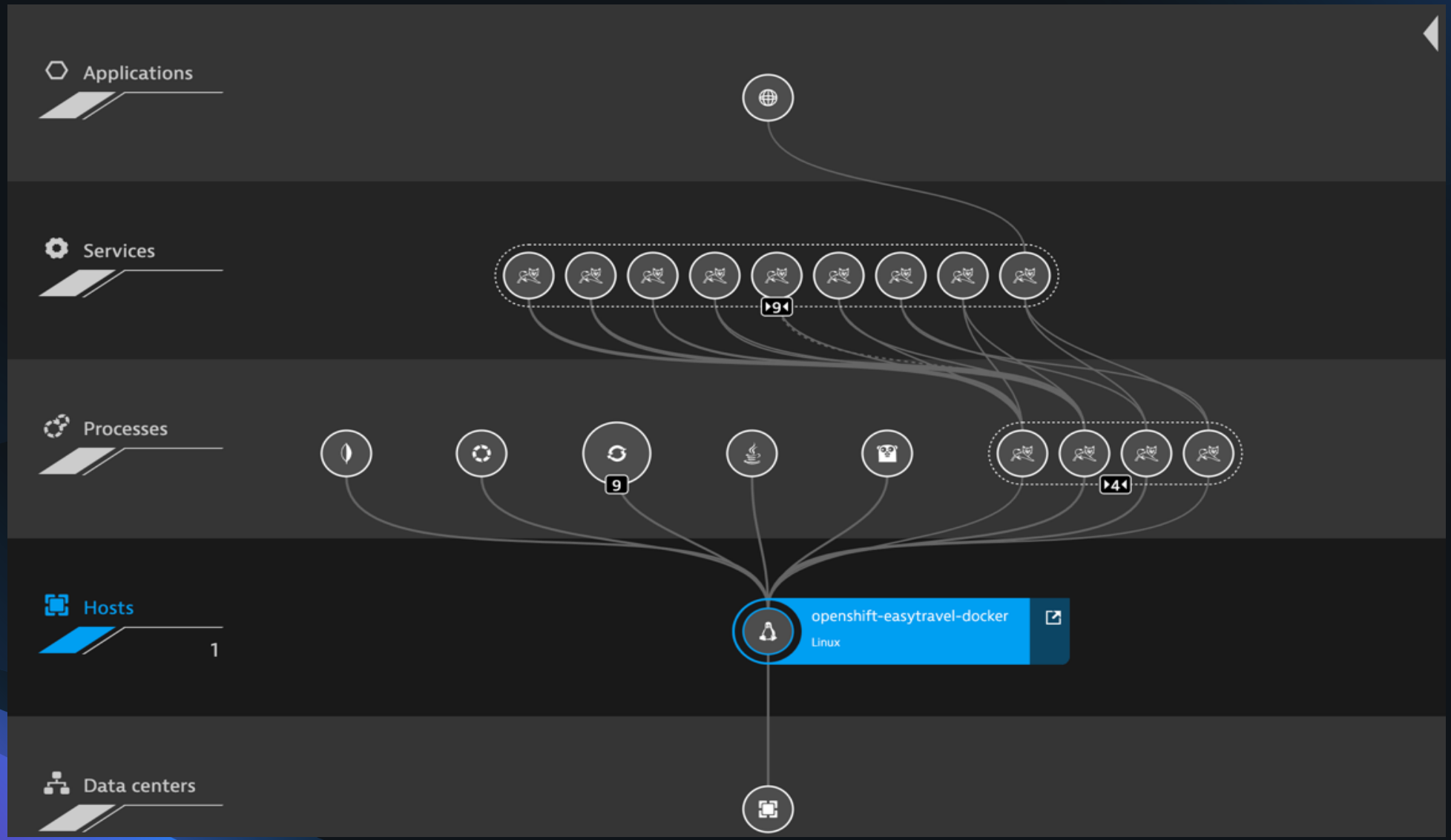
Exercise #2
Deploy the OneAgent Operator

Out-of-the-box Kubernetes/OpenShift monitoring

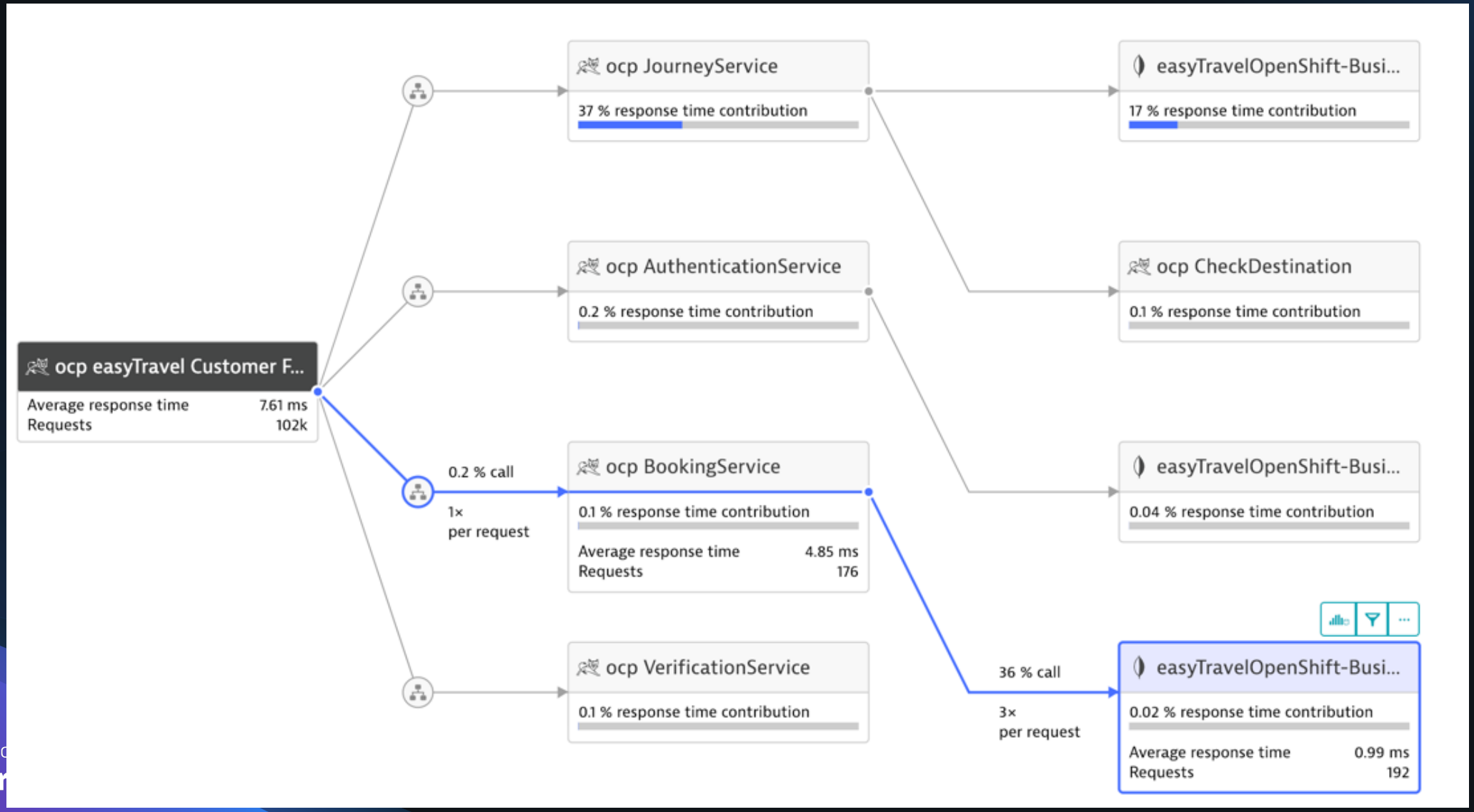
What you get ootb with Dynatrace for k8s/ocp?

- Automatic agent management by Operator
  - Every worker node have an agent deployed via daemonset
  - Automatic agent update
  - Automatic discovery : no image or container manipulation required
  - Automated full stack monitoring: hosts, containers, processes, services, application
  - Automatic capture of container and pod metadata
- Automated transaction tracing
- Container log monitoring

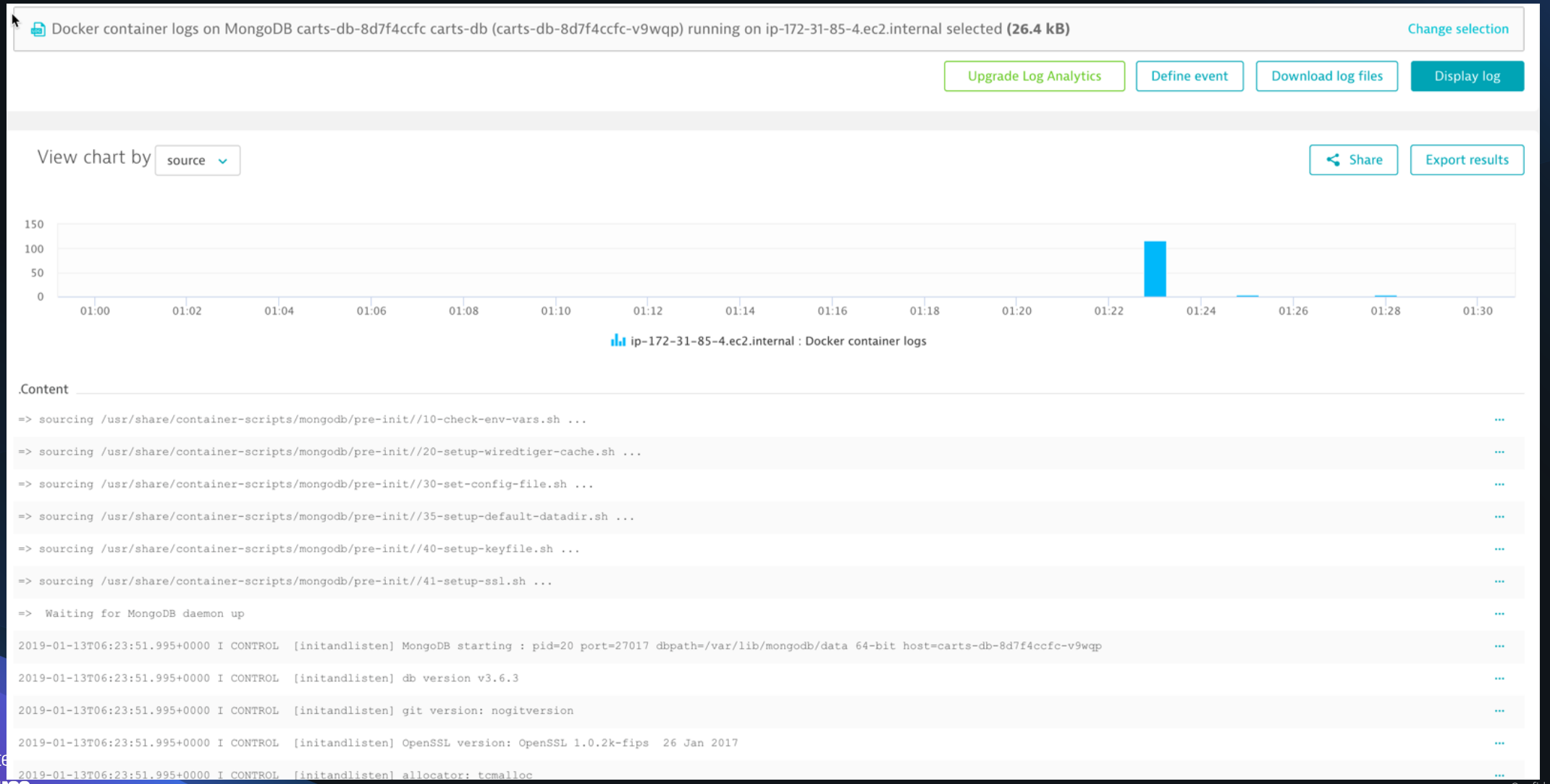
Automatic discovery and fullstack monitoring



Automatic transaction tracing

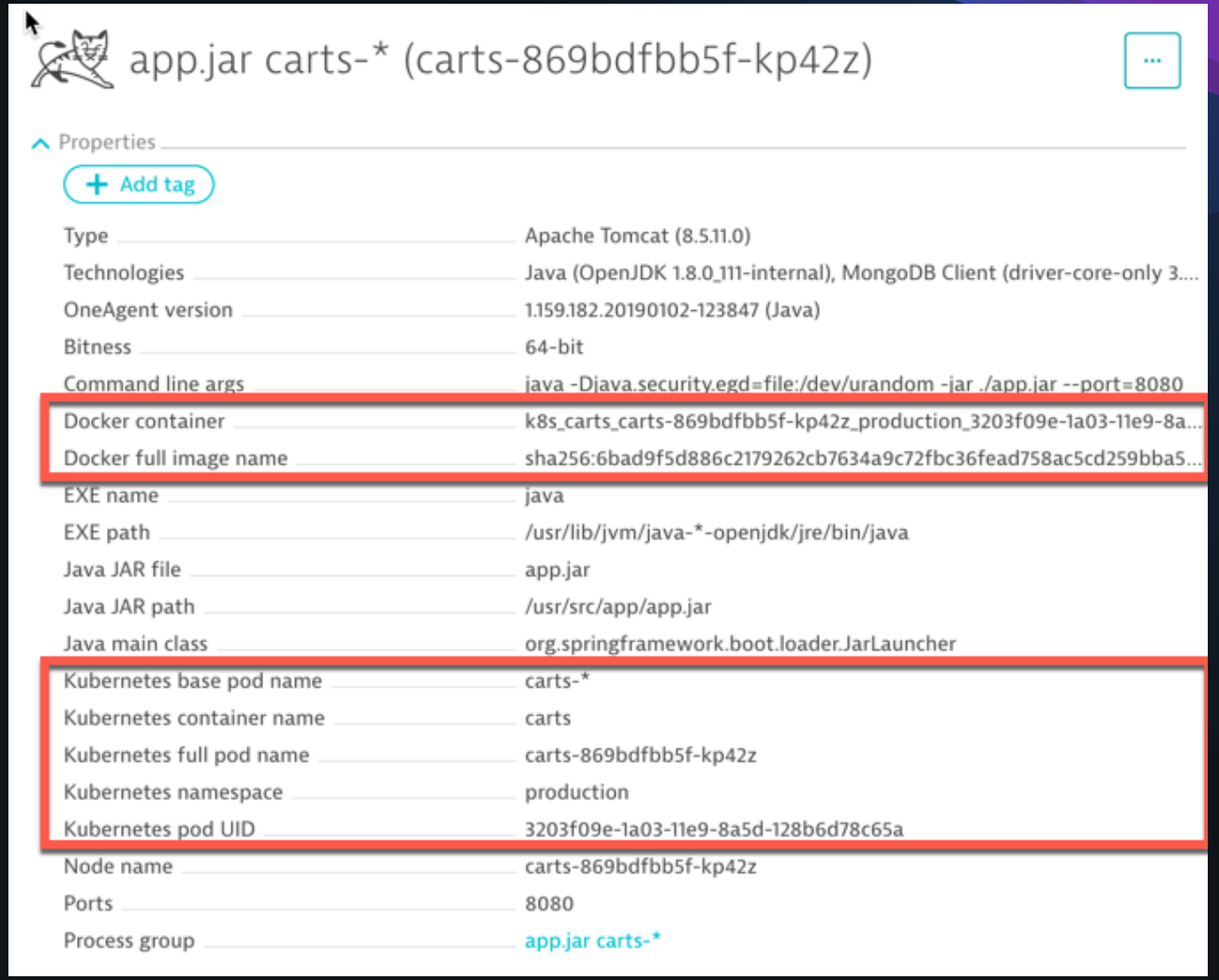


Container log monitoring



Automatic container and pod metadata

- Docker container name
- Docker container image name
- **Kubernetes base pod name**: User-provided name of the pod the container belongs to
- **Kubernetes container**: Name of the container that runs the process
- **Kubernetes full pod name**: Full name of the pod the container belongs to
- **Kubernetes namespace**: Namespace to which the containerized process is assigned
- **Kubernetes pod UID**: Unique ID of the related pod.



Automation rules to leverage Kubernetes context

Kubernetes metadata : Labels

- Key/value pairs that are attached to objects, such as pods
- Used to specify identifying attributes of objects that are meaningful and relevant to users and organizations
- Enable users to map their own organizational structures onto system objects in a loosely coupled fashion
- Via a label selector, the client/user can identify a set of objects in Kubernetes

Kubernetes metadata : Annotations

- Used to attach arbitrary metadata to Kubernetes objects.
- Metadata can be structured or unstructured, large or small.
- Unlike labels, annotations are not used to identify or select objects

Kubernetes metadata import in Dynatrace

- Ideally, you want to avoid duplicating selectors, filters, metadata across multiple tools having those uniquely defined in Kubernetes.
- Kubernetes can be configured to allow the OneAgent to import labels and annotations in Dynatrace, avoiding duplications.
- The OneAgent will use a pod service account to query for its metadata via the Kubernetes REST API.
- Kubernetes metadata can be useful in Dynatrace for filtering, structure and providing additional context. Management Zones, Alerting Profiles, API call filters, dashboard filters.

Exercise #3
Set up automatic import of Kubernetes labels and annotations

Additional context import mechanism

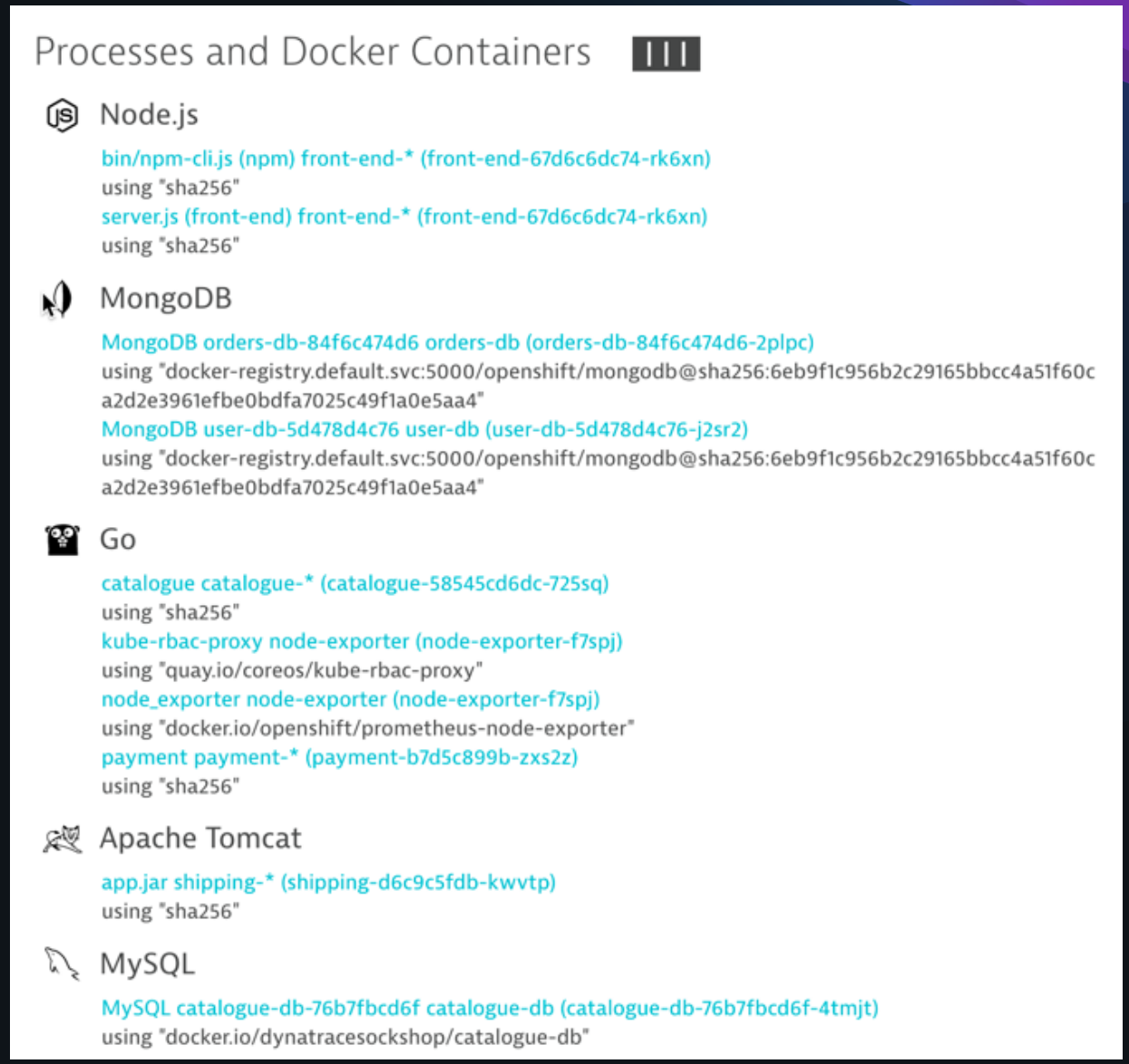
- Additional to the capability of importing Kubernetes pod labels and annotations, Dynatrace also offer the capability of defining tags and properties via process environment variables.
  - Tagging: Environment variable DT_TAGS
  - Can define multiple tags, either as strings or key/value pairs, separated by a space
    - Example: DT_TAGS=MyFirstTag MySecondTag=Tag2Value
  - Properties: Environment variable DT_CUSTOM_PROP
  - Can define multiple properties as key/value pairs, separated by a space
    - Example: DT_CUSTOM_PROP=MyProp1=Prop1Value MyProp2=Prop2Value

Process Group and Service Naming

- Depending on the technology, Dynatrace will apply out-of-the-box rules in a best effort to automatically name the processes, process groups and services that are discovered.
  - In some cases, the auto-generated names may either be too generic or not reflect your naming standards
  - Also, in large enterprise multi-platform and multi-application environments, you can easily get lost without a good naming convention.

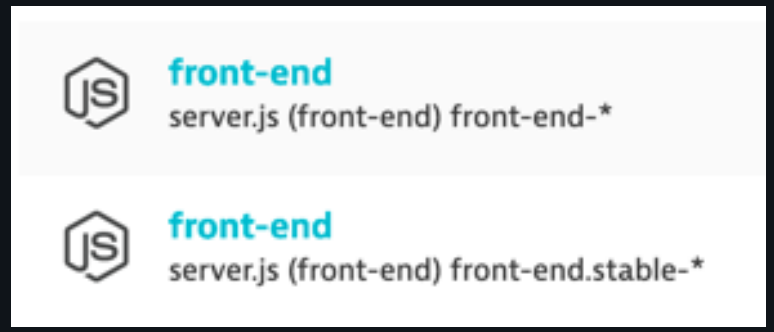
Process Group Naming

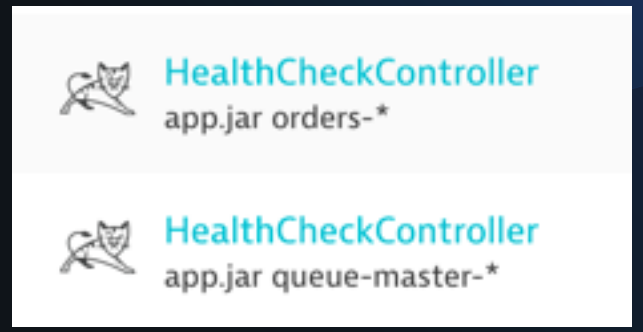
- With these process names, how do you know:
  - Which process is running in a pod on Kubernetes?
  - Which one is not, outside of Kubernetes?
  - Which one is part of the application or part of the Kubernetes platform?
  - Which namespace?
  - Where is my container name?
  - Which one runs in production or in dev?
  - Which one is my canary release vs my stable release?

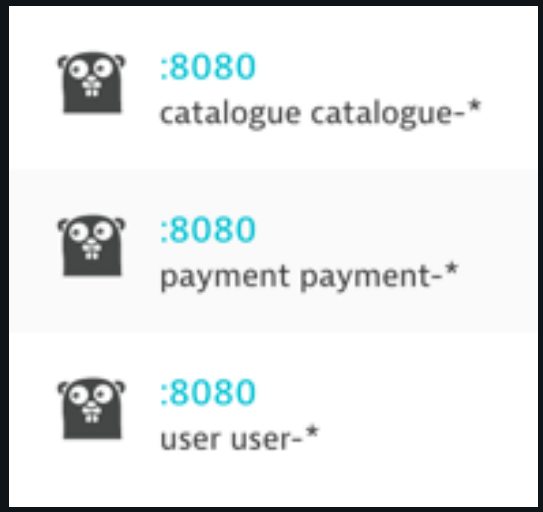


Service Naming

- Same for Services:
  - Which service is running in a pod on Kubernetes?
  - Which one is not, outside of Kubernetes?
  - Which one is production? Which one is dev?
  - I have duplicate names even though they are different services
  - What can I do with a service called ":8080"?







Customize Process Group and Service Naming

- In Dynatrace, you can define process group naming rules
  - Applied automatically, no restart needed
  - Don't affect how processes and services are discovered and grouped
  - Can be modified or deleted without any data loss
  - Naming rules can leverage process and service metadata to enforce granular and contextual naming standards
- Recommended best practice for k8s/ocp

Exercise #4
Customize Process Group naming rules for Kubernetes/OpenShift

Exercise #5
Customize Service naming rules for Kubernetes/OpenShift

Organize your Kubernetes monitoring entities with Management Zones

Why Management Zones for Kubernetes monitoring?

- It is common that Kubernetes clusters are infrastructure shared by multiple delivery stages, projects or lines of business
  - Scale and complexity of the moving pieces can make it challenging to find what you want to focus on
  - You want also to be able to control access and permissions
- Management Zones allow to set up flexible and sophisticated rules filter and segregate data and access
  - Per namespace
  - Per label and/or annotation
  - Per application
  - App-level data vs infrastructure-only

Management Zones : What it is, what it is not

- What it is :
  - A concept to group parts of an environment together
  - Each entity can be part of any number of management zones
  - It allows users to focus on one management zone by filtering all views
  - Dashboards, list views and Smartscape can be filtered (global search)
  - Fine grained role-based access control
  - Access rights can be granted either to a full environment or to a number of Management Zones
- What it is NOT : a mechanism to apply configuration to a set of entities

Exercise #6
Management Zone by application

Exercise #7
Management Zone by namespace

Exercise #8
Role-Based Access Control with Management Zones

Custom alert notifications

Exercise #9
Setup alert notifications using Alerting Profiles

Kubernetes cluster health monitoring

Why monitoring Kubernetes clusters?

- It is common that Kubernetes clusters are infrastructure shared by multiple delivery stages, projects or lines of business
- Cluster administrators are responsible for providing enough resources and capacity to properly host and run project workloads
- They need to understand:
  - Cluster health and utilization of nodes
  - Health status of individual nodes
  - Actual usage of resources and how much resources has been requested
  - How much additional workload can be deployed per node

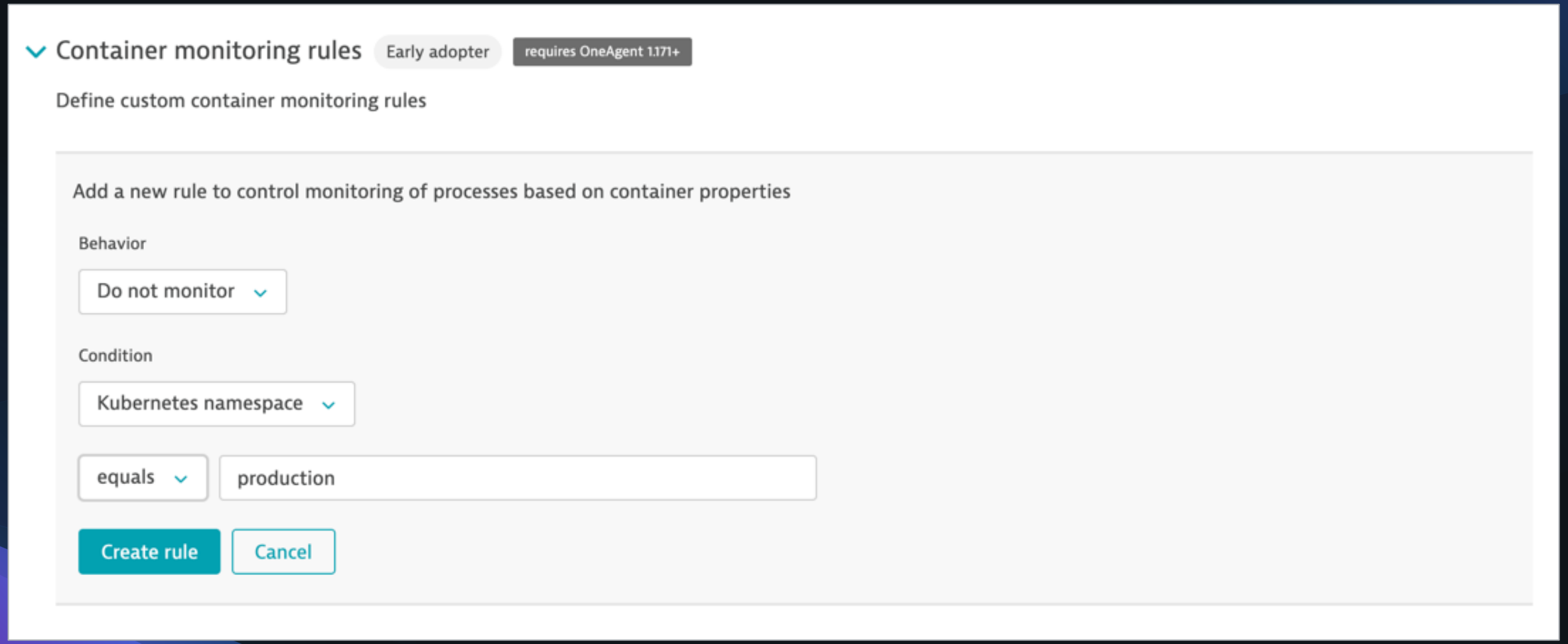
Exercise #10
Configure Kubernetes/OpenShift cluster integration

Agent lifecycle management

Automatic agent upgrade via the Operator

Misc

Container Monitoring Rules

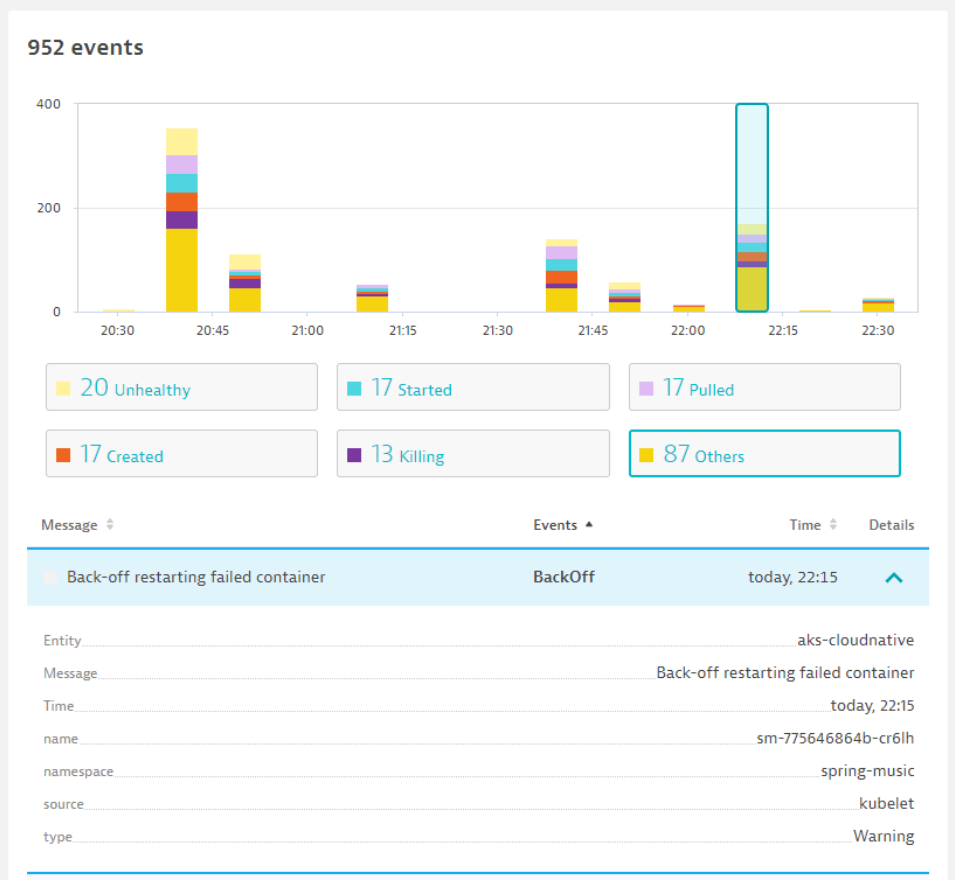


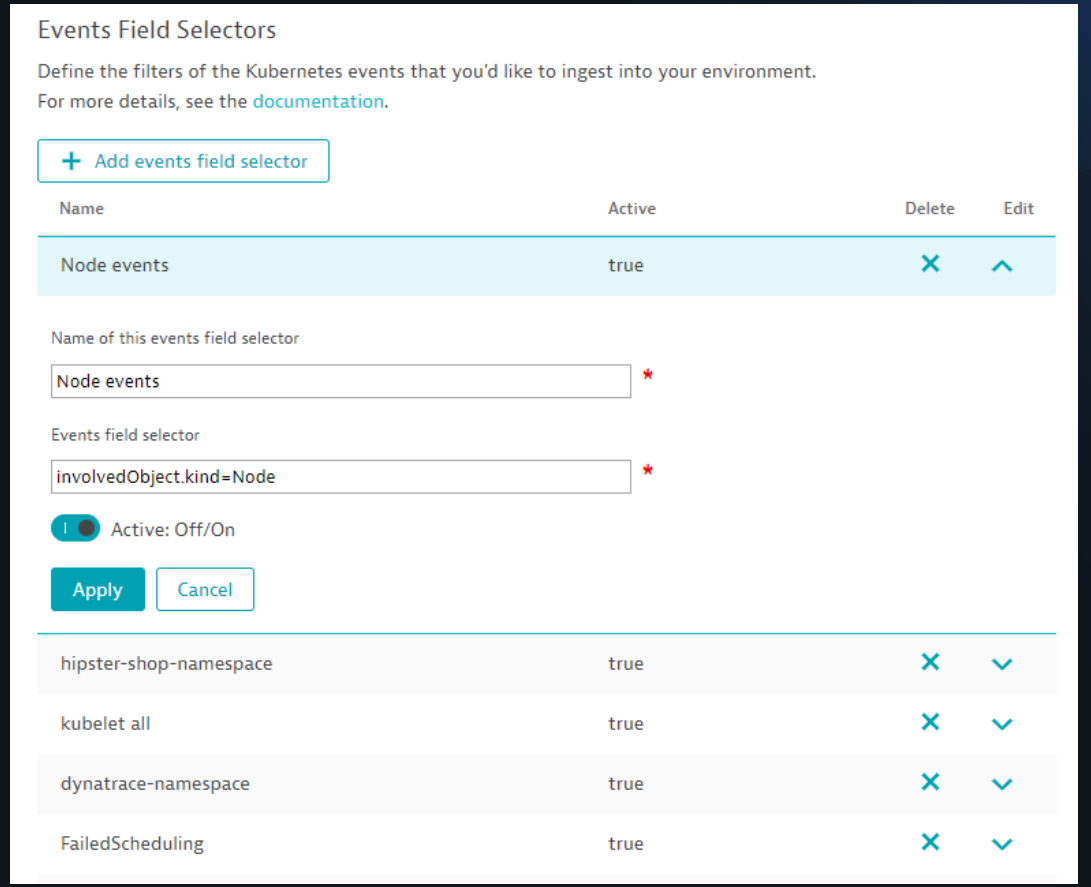
Mixed (infra + fullstack) mode

- Customer wants to monitor some of its cluster node in Infrastructure mode and the other in Full Stack mode
- How do you do that with the Operator?
  - Essentially, you need one Operator
  - You need to leverage node labels (existing or create ones)
  - You can use the same secret or create two secrets (one for full-stack, one for infra)
  - You will need 2 custom resources (cr.yaml) definitions
    - Each respective cr.yaml will use the appropriate node selectors
    - The infra cr.yaml will use the installer argument INFRA_ONLY=1

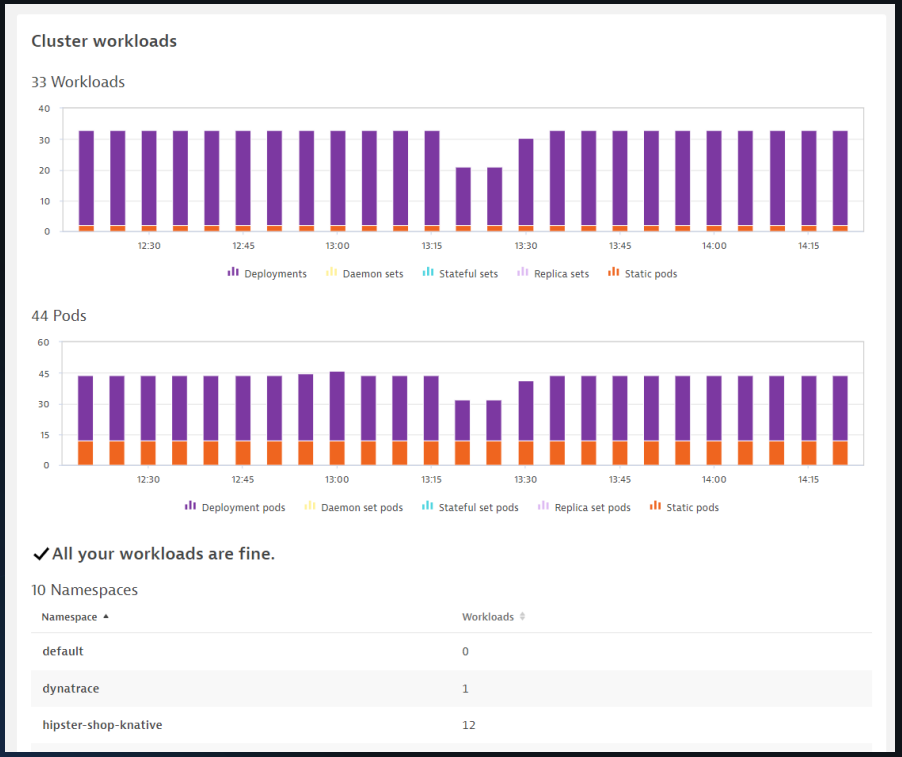
What's coming?

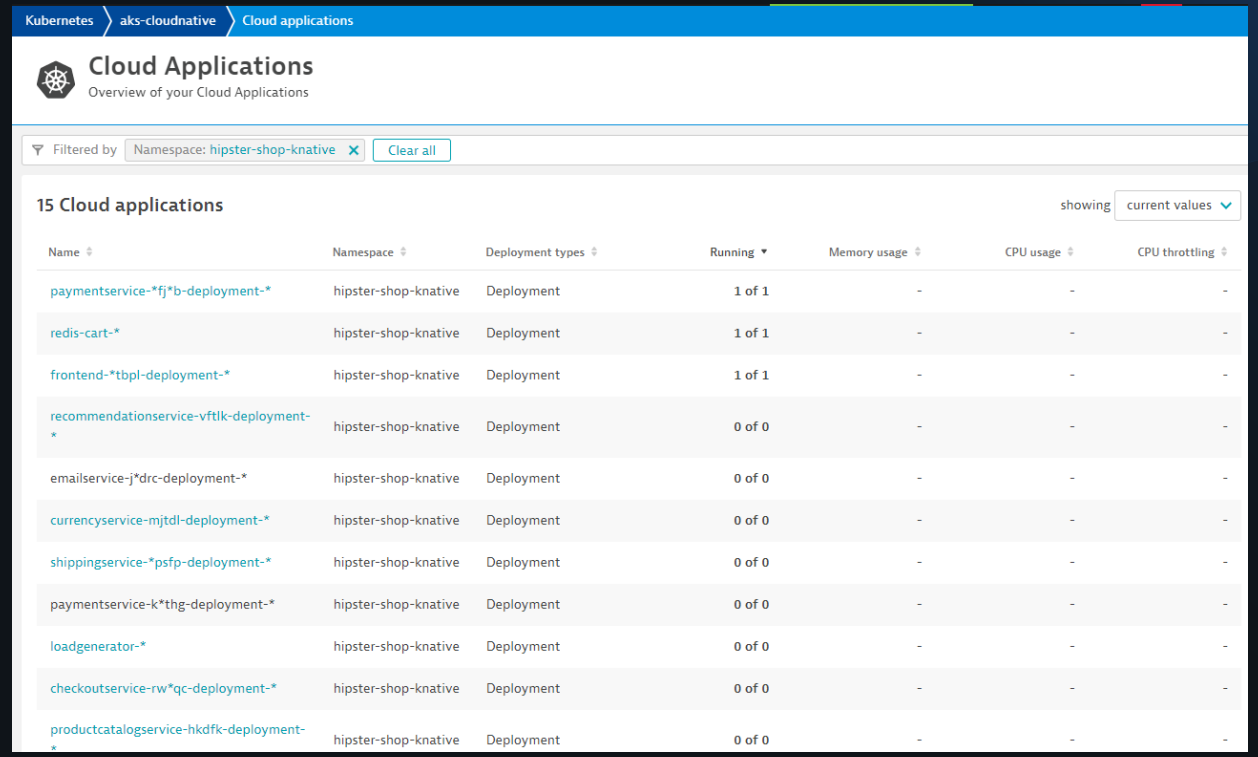
Event ingestion





Cloud application view





More on what's coming?
Don't miss this breakout session:

Doubling down on k8s with Dynatrace
Thursday 2pm Nolita 1-2

dynatrace.com