RED HAT
FORUM
Europe, Middle East & Africa

Technical Aspects of Workload Containerization

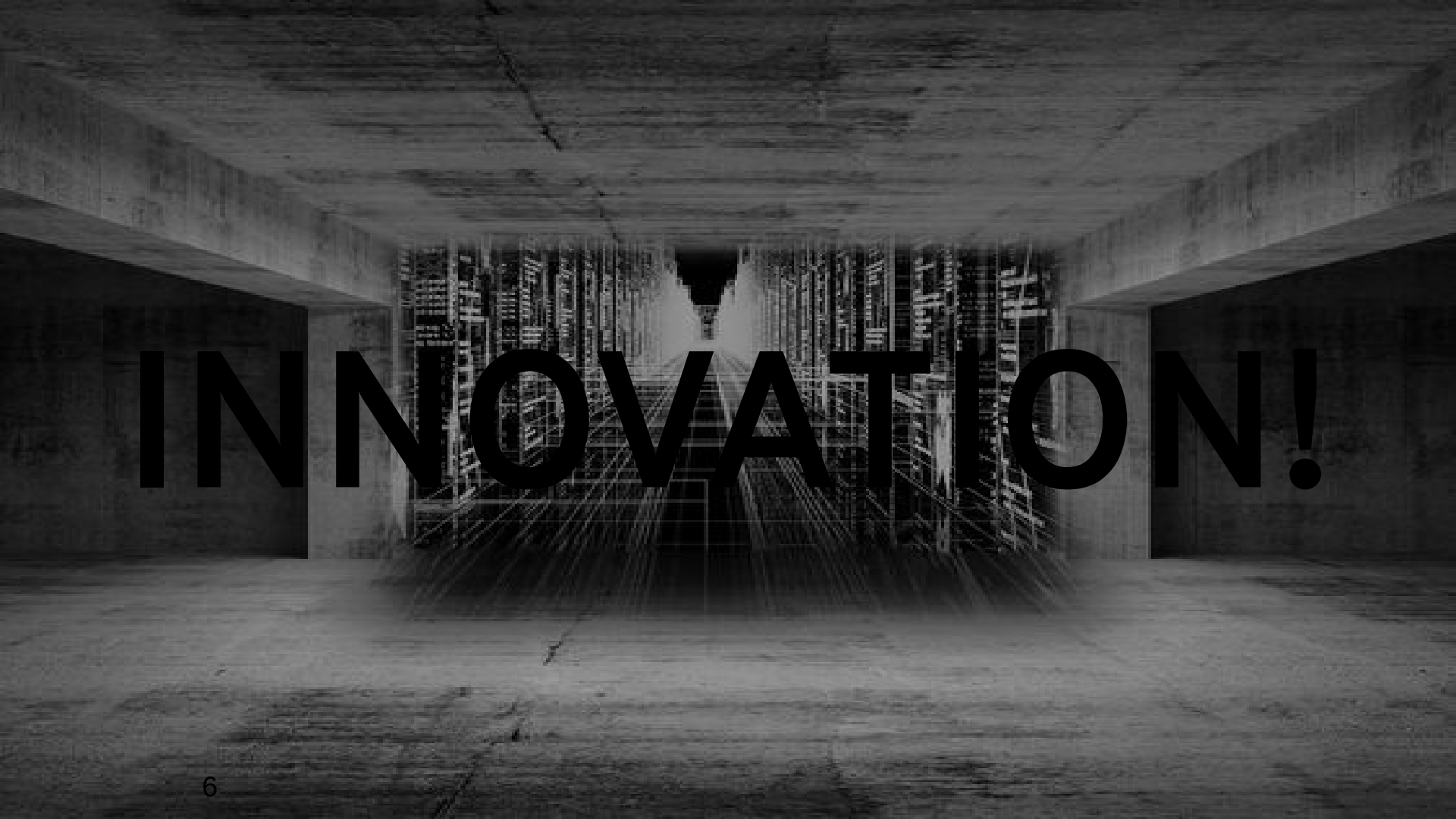
Daniel Fröhlich
DACH Solution Architect

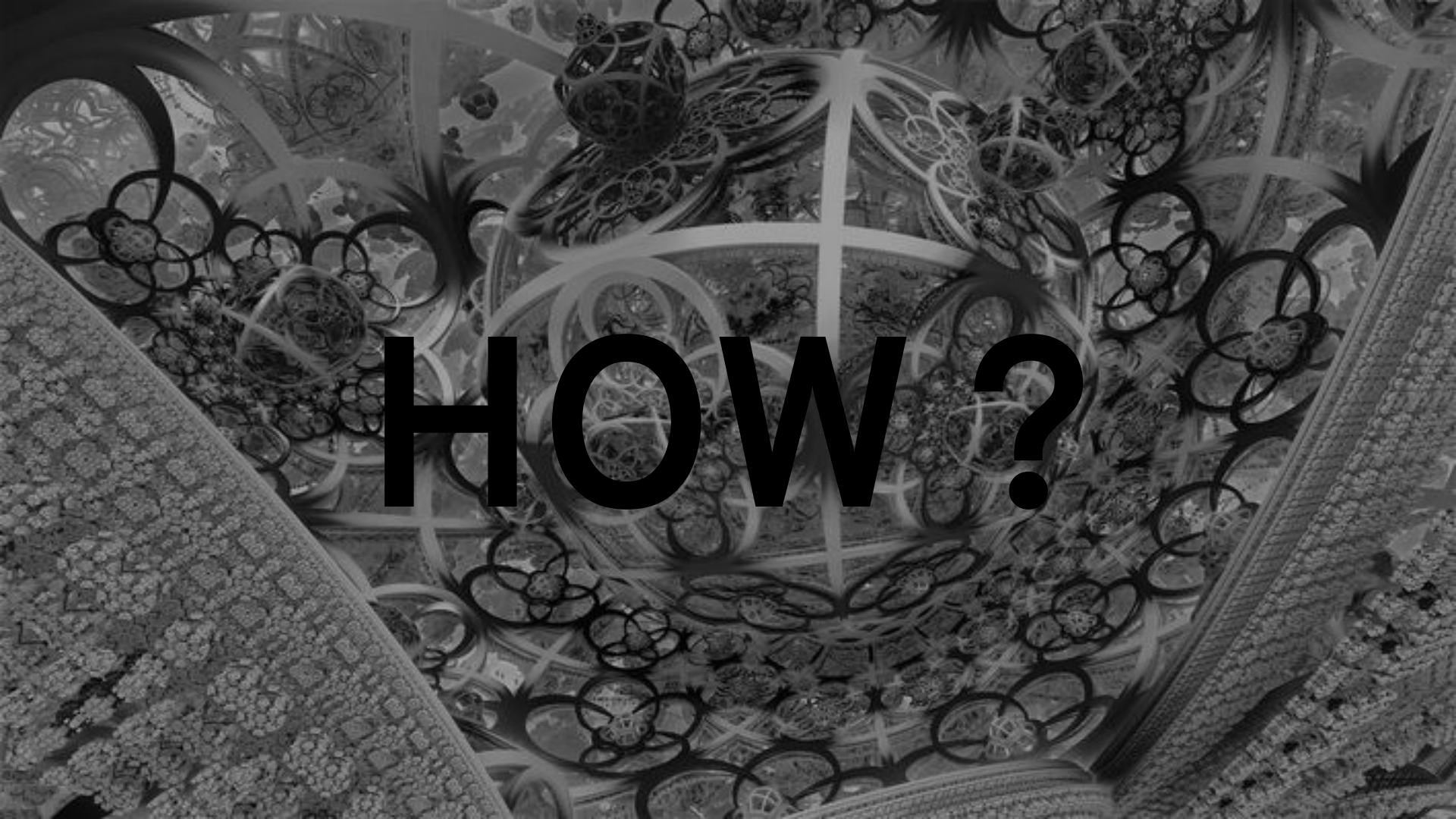

CROWLEY
CROWLEY


CROWLEY
WHY?


SPEED!

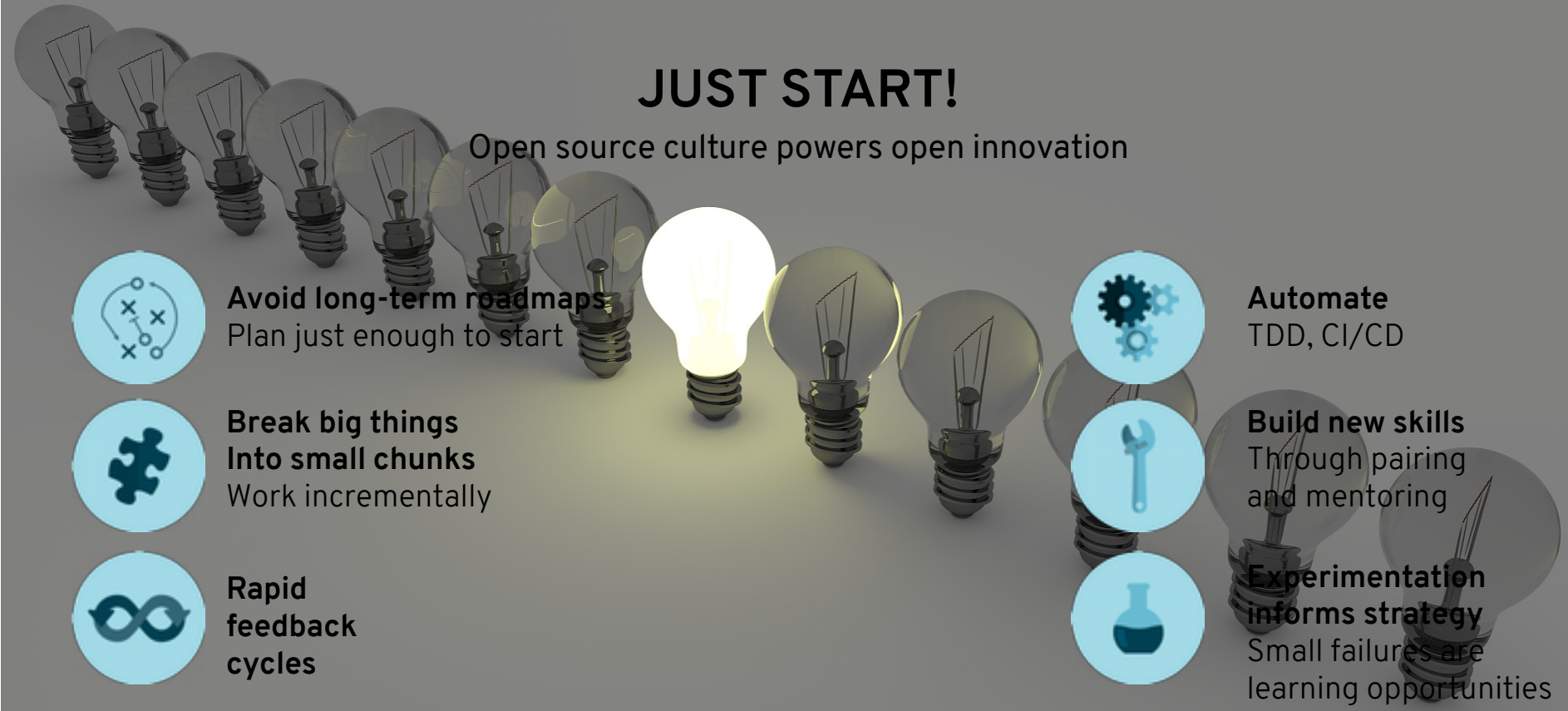

COST!


INNOVATION!


HOW ?

JUST START!

Open source culture powers open innovation



Avoid long-term roadmaps
Plan just enough to start

Break big things Into small chunks
Work incrementally

Rapid feedback cycles

Automate
TDD, CI/CD

Build new skills
Through pairing and mentoring

Experimentation informs strategy
Small failures are learning opportunities

MIGRATING WORKLOADS TO CONTAINERS

BUILD MIGRATION CASE

We help calculate and analyze ROI based on:

TECHNICAL:
Application scalability
Auto deployment
Infrastructure agnostic

BUSINESS:
Consolidation
Flexibility
Speed to market

DETERMINE FIT AND PRIORITY

We help prioritize migrations based on:

PORTABILITY:
Ease of switching servers
Ease of switching cloud providers

SCALABILITY:
Ability to run multiple instances

ACTIVE DEVELOPMENT:
Ability to make changes

PLAN MIGRATION

We help plan migration activity:

CHANGE CODE:
Required: fix incompatibilities
Value add: architect for cloud

AUTOMATE BUILD:
Required: S21/ Docker script
Recommended: CI/CD

CUSTOMIZE IMAGE:
Required: support development languages and libraries
Value add: support shared libraries and standardized configuration

EXECUTE MIGRATION

We help execute migrations through an iterative process:

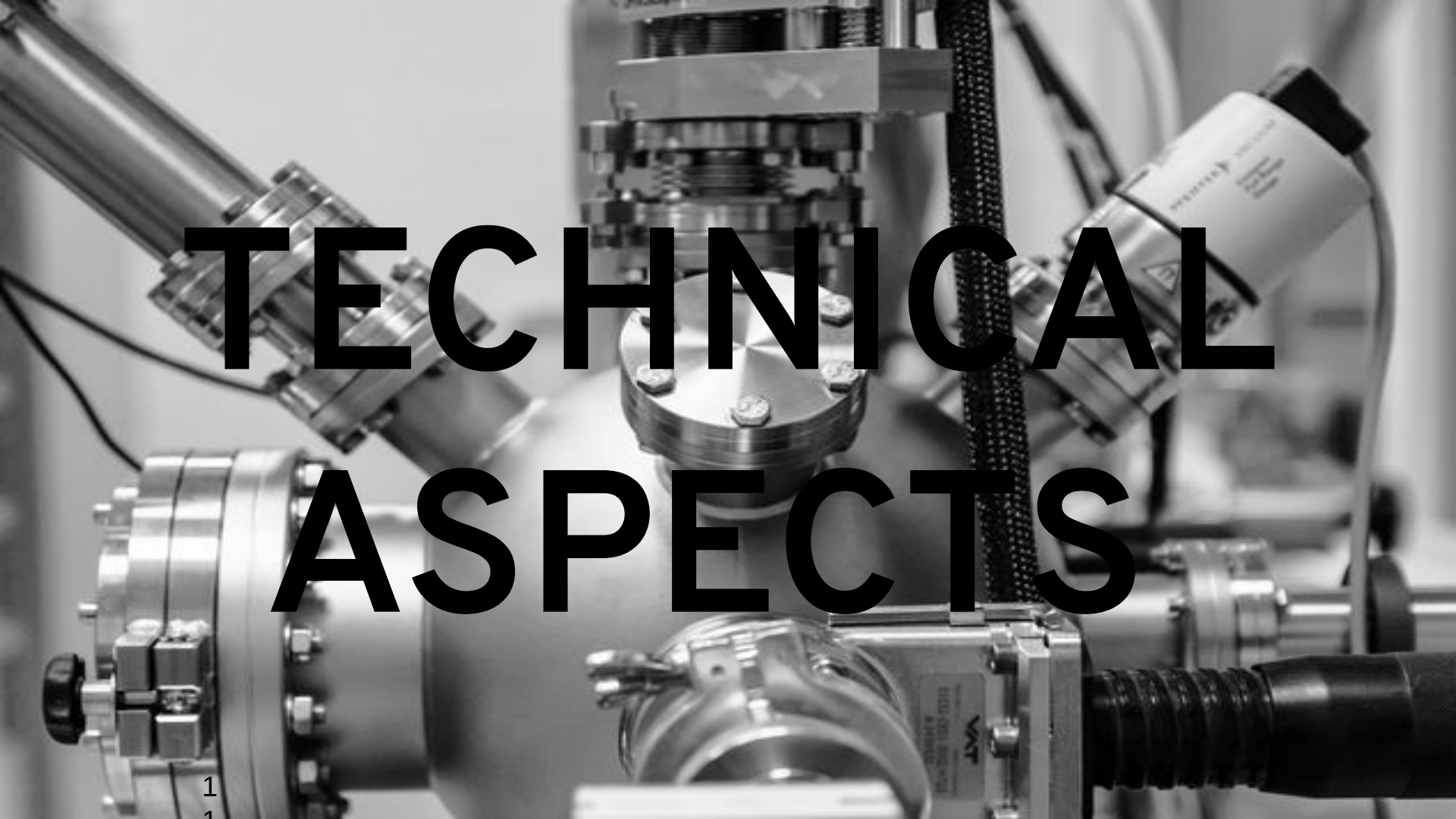
1. Perform all required tasks
2. Perform all recommended tasks
3. Assess and perform "value add" tasks individually based on need

...and **emphasize continuous improvement** to drive down incremental migration costs over time.

EXECUTE MIGRATION BY MOVING...

1: TO RHEL	2: TO DOCKER	3: TO LOCAL OCP	4: TO TARGET OCP
Use RHEL for Developers (provides no-cost subscriptions for development use only)	Getting started with containers OpenShift Developer Guide Docker Best Practises	Use Container Development Kit on your laptop Expect Problems: with Security	Define CI/CD Pipeline Expect Problems with Permissions, Operator Limitations

INVOLVE EXPERT GUIDANCE



TECHNICAL ASPECTS


App has complex and lengthy startup procedures

Feature(s): Init Containers

Description: Init containers provide a chained, pre-deployment mechanism for running arbitrary containers/programs as part of getting ready for your application to run. They aid in application deployment (pre-check, pre-start coordination, data population, etc) and, to some degree, dependency management.
How it Works: Init containers run to completion and each container must finish before the next one starts, and will run for each pod instance. The init containers will honor the restart policy. Leverage initContainers in the podspec.

```
$ cat init-containers.yaml
apiVersion: v1
kind: Pod
metadata:
  name: init-loop
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: workdir
      mountPath: /usr/share/nginx/html
  initContainers:
  - name: init
    image: centos:centos7
    command:
    - /bin/bash
    - "-c"
    - "while :; do sleep 2; echo hello init container; done"
  volumes:
  - name: workdir
    emptyDir: {}
```

```
$ oc get -f init-containers.yaml
NAME      READY    STATUS    RESTARTS   AGE
nginx   0/1      Init:0/1   0         6m
```


Apps need persistent storage

Feature(s): [OCP 3.6 + CNS 3.6 (Sep 2017)]

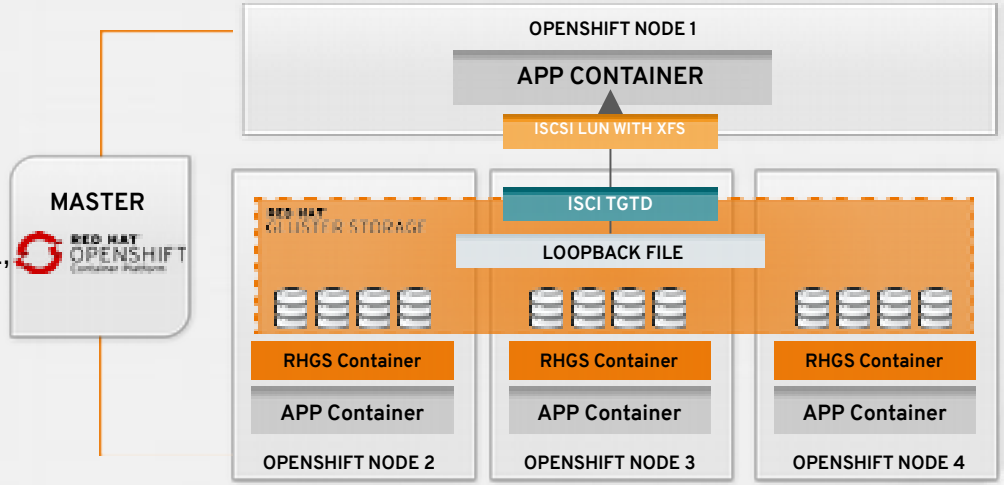
Description: CNS 3.6 (Sep 2017) with OCP 3.6 will support block, higher volume density & S3 object store

How it Works:

- RWO volumes backed by Gluster-iSCSI block storage provide better performance for MySQL, PostgreSQL, etc. and targeting support for Elastic Search (OCP Logging)

- 3x volume density per cluster (1000+) with lower memory footprint of RHGS (Brick-mux)

- S3 object service for OCP (based on swift-on-file, *Tech Preview*)




Security Team is missing
firewall between app server
and database

Tech Preview

Feature(s): Network Policy

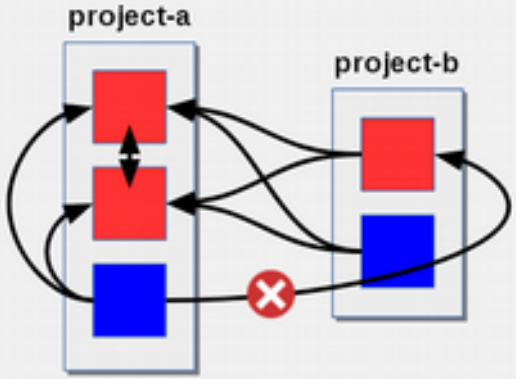
Description: Optional plugin specification of how selections of pods are allowed to communicate with each other and other network endpoints.

How it Works: Fine-grained network namespace isolation using labels and port specifications

- what ingress traffic is allowed to any pod, from any other pod
- on specific ports
- including traffic from pods located in other projects

Additional Enhancements for 3.6:

- Make services work with NetworkPolicy
- Implement NetworkPolicy support with PodSelectors (v1)
- Implement NetworkPolicy support for specific ports



Policy applied to namespace: **project-a**

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-to-red
spec:
  podSelector:
    matchLabels:
      type: red
  ingress:
  - {}
```

Networking

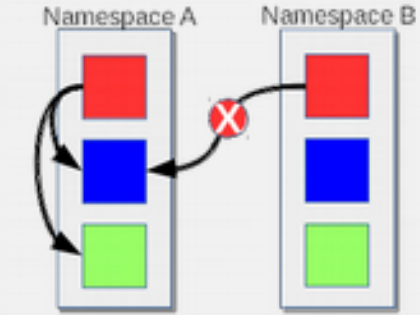
Feature(s): Network Policy (tech preview)

Description: Plugin (optional) specification of how selections of pods are allowed to communicate with each other and other network endpoints.

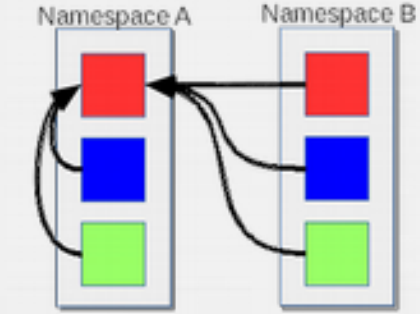
How it Works: Namespace isolation at the network layer using defined labels. Can also limit connections to specific ports (e.g. only TCP ports 80 and 443).

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-http-and-https
spec:
podSelector:
ingress:
- ports:
  - protocol: TCP
    - port: 80
    - port: 443
```

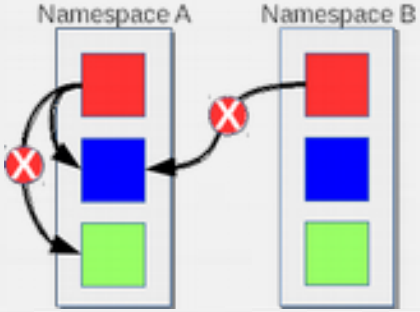
NOTE: Designed to provide an early look for testing/development. Not all features will be available.




```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-from-red
spec:
  podSelector:
  ingress:
  - from:
    - podSelector:
        matchLabels:
          type: red
```




```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-to-red
spec:
  podSelector:
    matchLabels:
      type: red
  ingress:
  - {}
```




```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-from-red-to-blue
spec:
  podSelector:
    matchLabels:
      type: blue
  ingress:
  - from:
    - podSelector:
        matchLabels:
          type: red
```


App needs ultra fast communication between components by using shared memory

Feature(s):
Shared Memory for Containers within the same pod

Description: When you decide to run more than 1 container in the same pod, it is likely that you would want them to be able to share memory segments. We now allow it in the default configuration.
How it Works: /dev/shm is shared across containers in a pod and is limited to 64mb and charged under the memcg cgroup.

```
$ kubectl create -f shared-memory.yaml
$ kubectl exec -it hello-openshift -c nginx /bin/bash
# echo "Hi" > /dev/shm/hi
$ kubectl exec -it hello-openshift -c mongo
/bin/bash
# cat /dev/shm/hi
Hi
```


App does not scale horizontally and needs active/passive hot standbys for high availability

Tech Preview

Feature(s): StatefulSets

Description: This new controller allows for the deployment of applications types that require changes to their configuration or deployment count (instances) to be done in a specific and ordered manner.

How it Works: StatefulSet offer more control over scale, network naming, handling of PVs, and deployment sequencing. They are in beta upstream in kube and therefore have reached tech preview in OCP 3.5. They will likely not come out of tech preview until OCP 3.8 or 3.9.

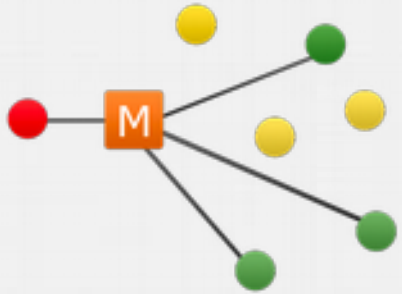
Supported:

1. Declaration of Ordinal Index
2. Stable Network ID nomenclature
3. Controlled/manual handling of PVs
4. Sequence Control at deployment time
5. Ordered control during scale up/down based on instance status

Not Supported:

6. Slow to iterate through the Ordinal Index and therefore slow on scale up/down
7. No deployment/pod spec post deployment verification of what is deployed Vs what is configured in the json.
8. Locality awareness of zones/regions when dealing with scale up/down ordinality changes or mounted PVs.


App DOES scale horizontally,
but needs multicast traffic
sync running instances



Feature(s): [Multicast Support]

Description: Multicast support so pods can send/receive traffic with other pods subscribed to the same multicast group.

How it Works:

requires ovs-multitenant plugin (or the new Network Policy plugin)

only annotated namespaces:

```
netnamespace.network.openshift.io/multicast-enabled: "true"
```

pods in different tenants can subscribe to same multicast group but can't see each others traffic

admin tenant (default project) multicast traffic does not appear in other projects

overlay (ovs+ tenants) and underlay (vm, physical server) multicast traffic never mix

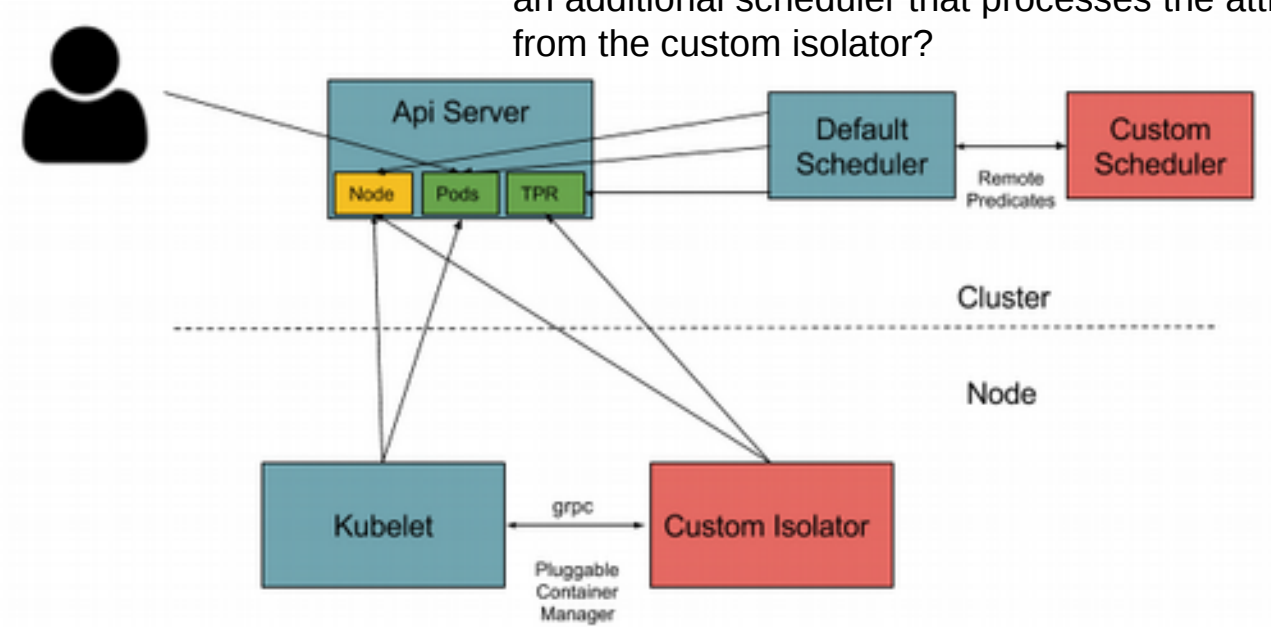
Caveat:

Best for low bandwidth coordination or service discovery -- not a high-bandwidth solution.


App requires GPU support
(image rendering, tensor flow)

GPU Decision Points

Should we use scheduler extensions to multi-schedulers to run an additional scheduler that processes the attributes coming from the custom isolator?



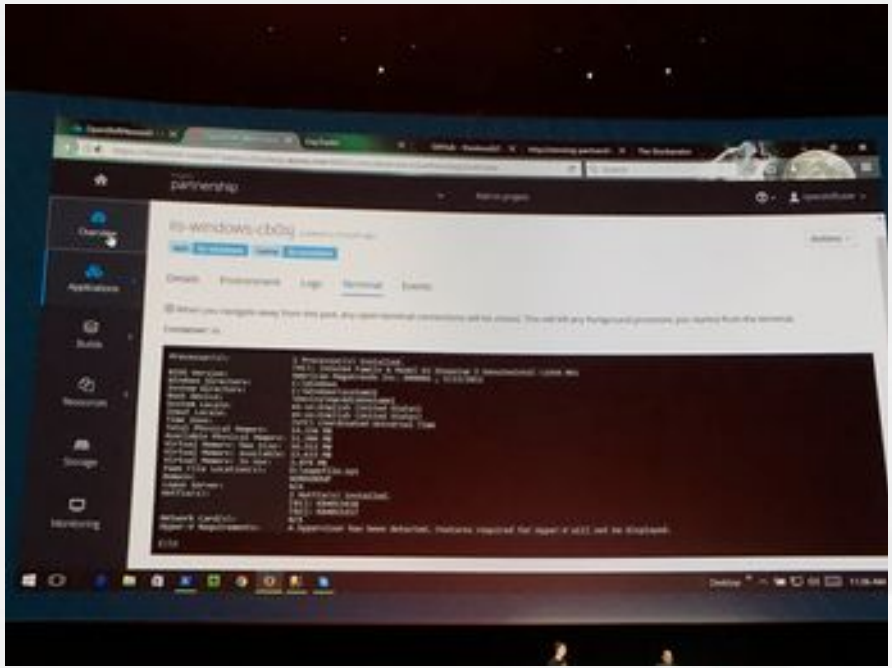
Should the kubelet be the function to load device drivers and kernel modules needed by the hardware or should the container runtime?

https://blog.openshift.com/use-gpus-openshift-kubernetes/


App requires Windows DLL

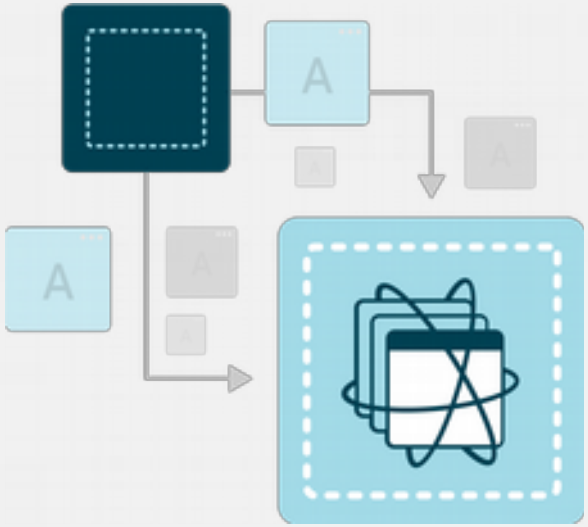
Windows Containers on OpenShift (Preview)

- Upstream project to improve Windows Containers and Kubernetes support
- **Collaboration between Microsoft, Red Hat and Kubernetes** (Windows SIG)
- Demo of OpenShift running mixed cluster with Windows Containers/Windows Server 2016 Nodes running with Linux Containers/RHEL Nodes
- Running in OpenShift (kube pod) running .NET 4.5. With a java container as well.
- Watch the demo video.




redhat

CONTAINERIZATION IN PRACTICE



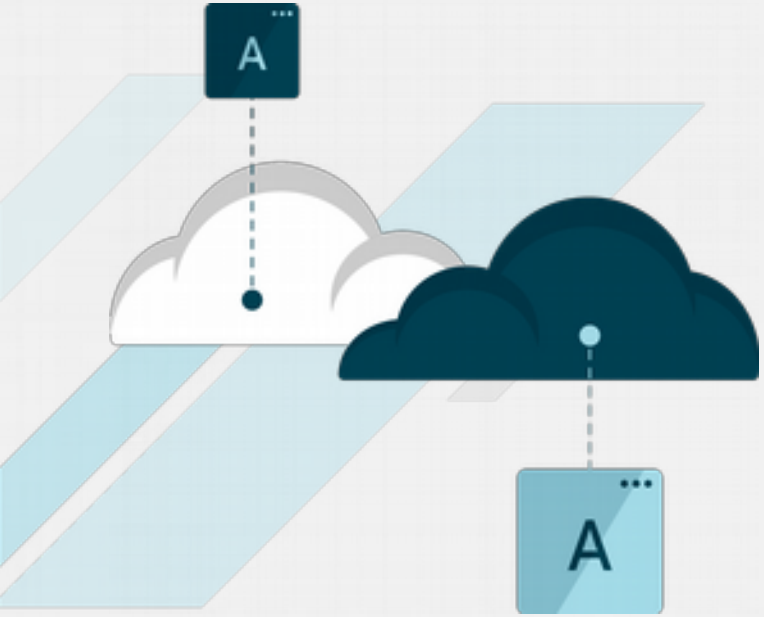
- **Conduct** high-level analysis
- **Address** major dependencies
- **Define** networking behavior
- **Stand up** application level control
- **Define** startup and runtime resourcing

CLOUD NATIVE APPLICATION IDEALS

Cloud native applications have some distinctive architectural characteristics
and are designed to leverage cloud environments.

- Ability to handle dynamic scaling, load configuration in flexible ways, and aggregate logs
- Dynamic discovery of dependencies
- Load balancing of requests to dependencies
- Enable application monitoring and distributed tracing
- Circuit breaker and bulkhead patterns
- Feature toggles and health checks

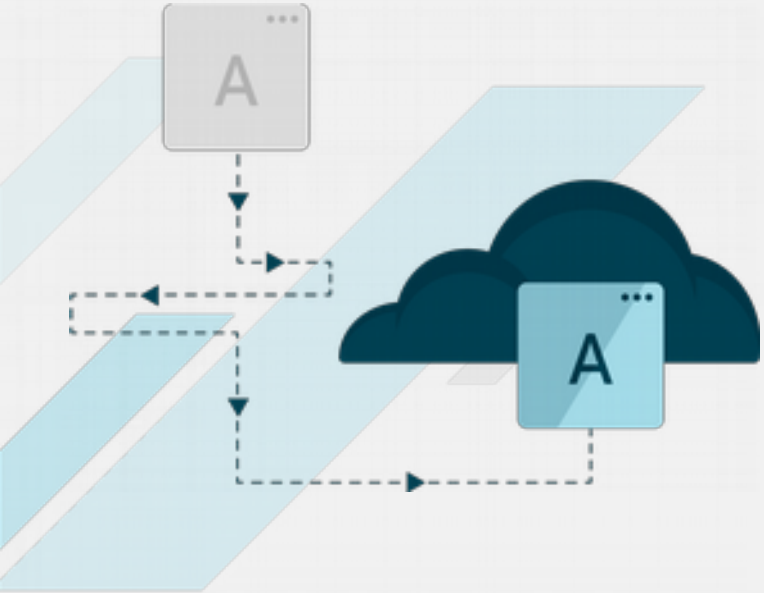
BEFORE GOING CLOUD NATIVE, START WITH CLOUD COMPATIBLE



We start with some simple application changes:

- Create support for external configurations
- Remove IP bindings
- Run on Red Hat® Enterprise Linux 6.4+ compatible libraries
- Ensure logs write to console/stdout

JOURNEY FROM CLOUD COMPATIBLE TO CLOUD NATIVE



Cloud compatible represents the **minimum viable product** required to onboard to container-based cloud platforms.

Progress can be made along the continuum towards cloud native with **subsequent iterations**.

Actual end state should be **dictated by the needs of the business**. Not every application needs to be fully cloud native in order to provide value!