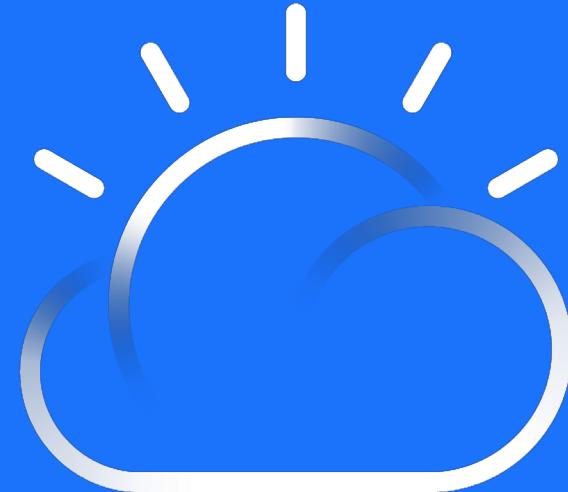


Scaling Integration

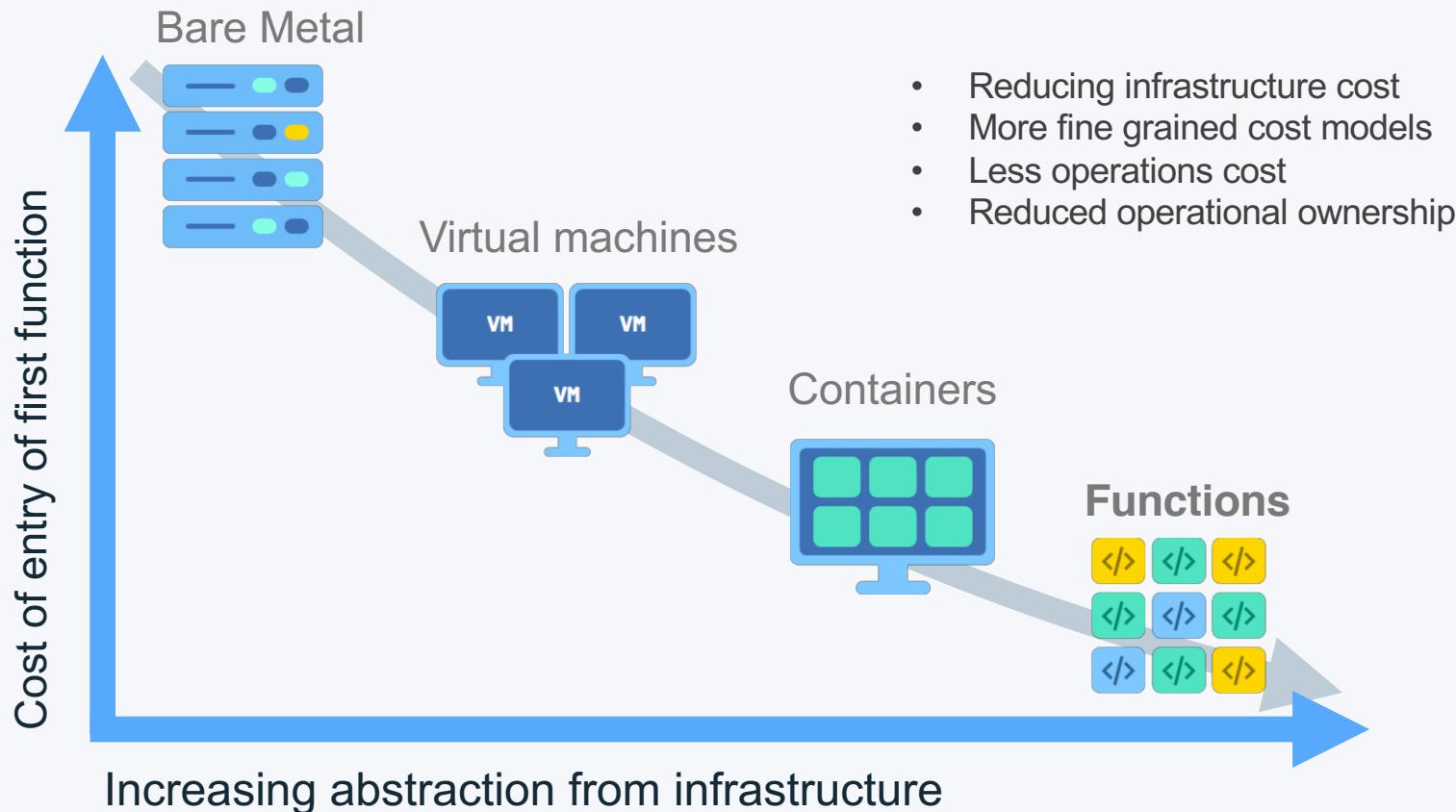
Kim Clark
Integration Architect

Rob Nicholson
Distinguished Engineer – Integration Portfolio



IBM Cloud

IBM



Benefits of a container based strategy

Build Agility

Team Productivity

Fine-grained Resilience

Scalability
and
Optimization

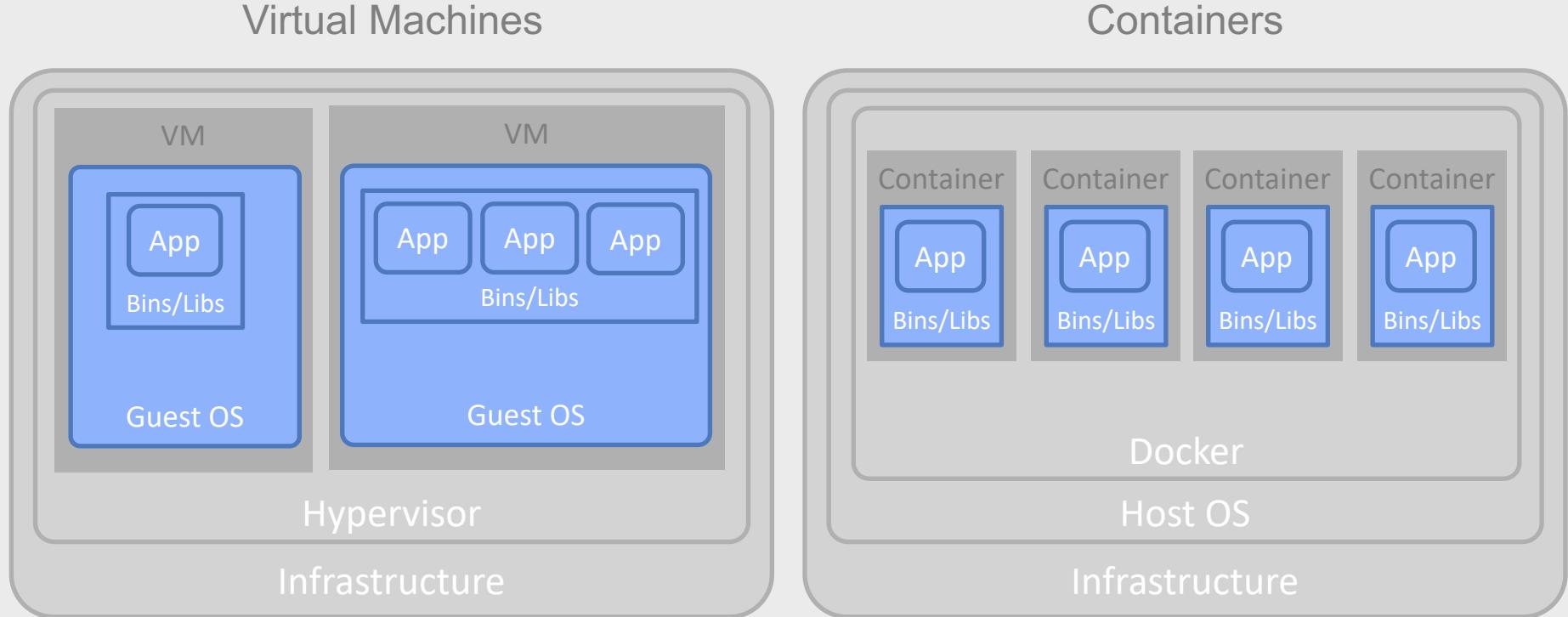
Operational Consistency

Component Portability

Focus of this presentation

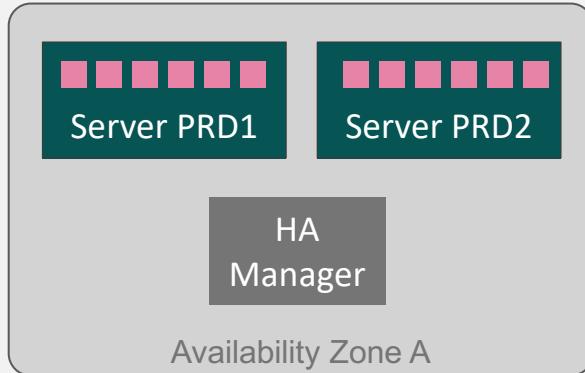
Containerization is more than just a re-platforming exercise. “lift and shift” will not bring the above benefits.
Requires: *Fine-grained deployment, organizational decentralization, pipeline automation, disposable components...*

Difference between virtual machines and containers



Derived from <https://www.docker.com/what-container>

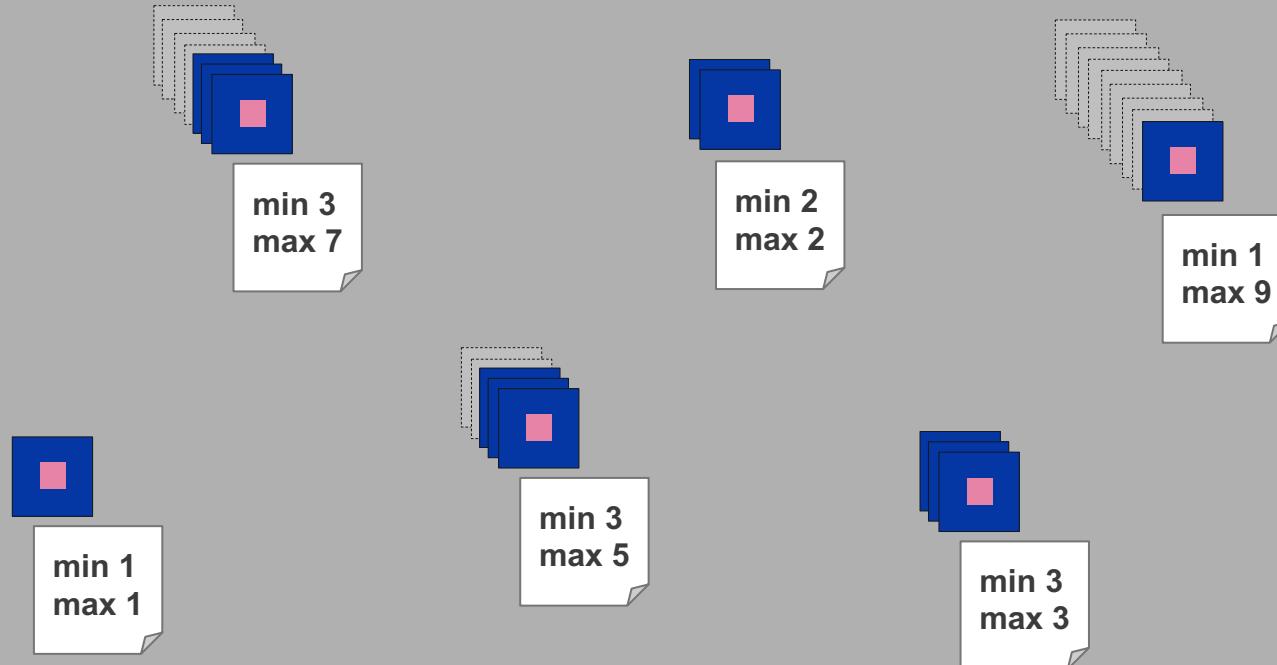
What's the container equivalent of the HA/DR topology you have today?



Traditional
(explicit configuration)

Container platform
(declarative logical configuration)

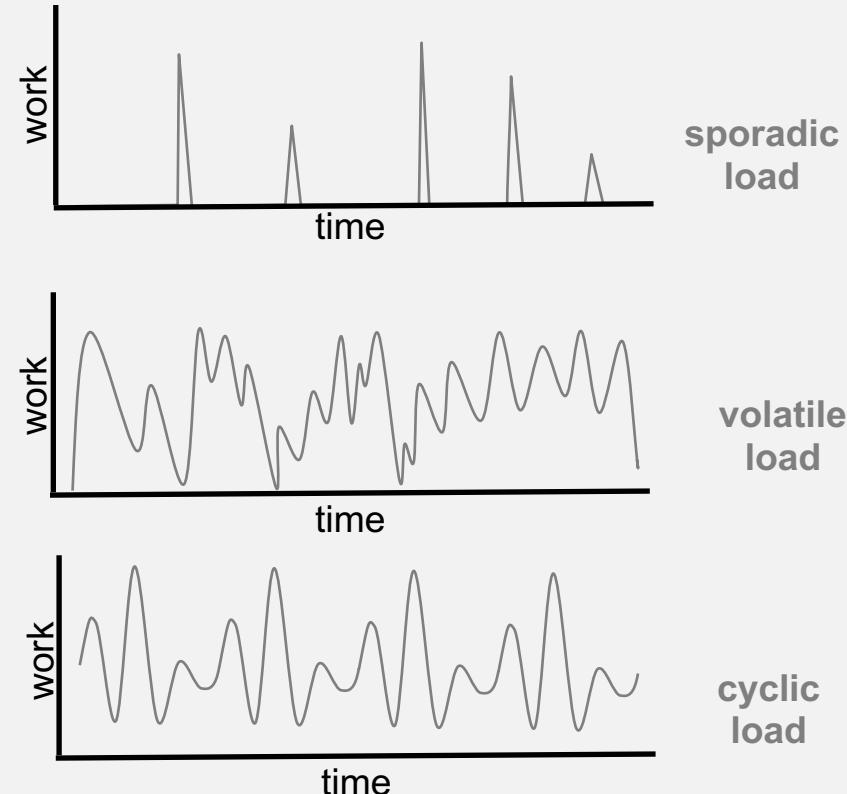
Containers enable fine-grained deployment...and discrete scaling policies
with near-complete abstraction from physical resources



Container orchestration platform (multi-zone)

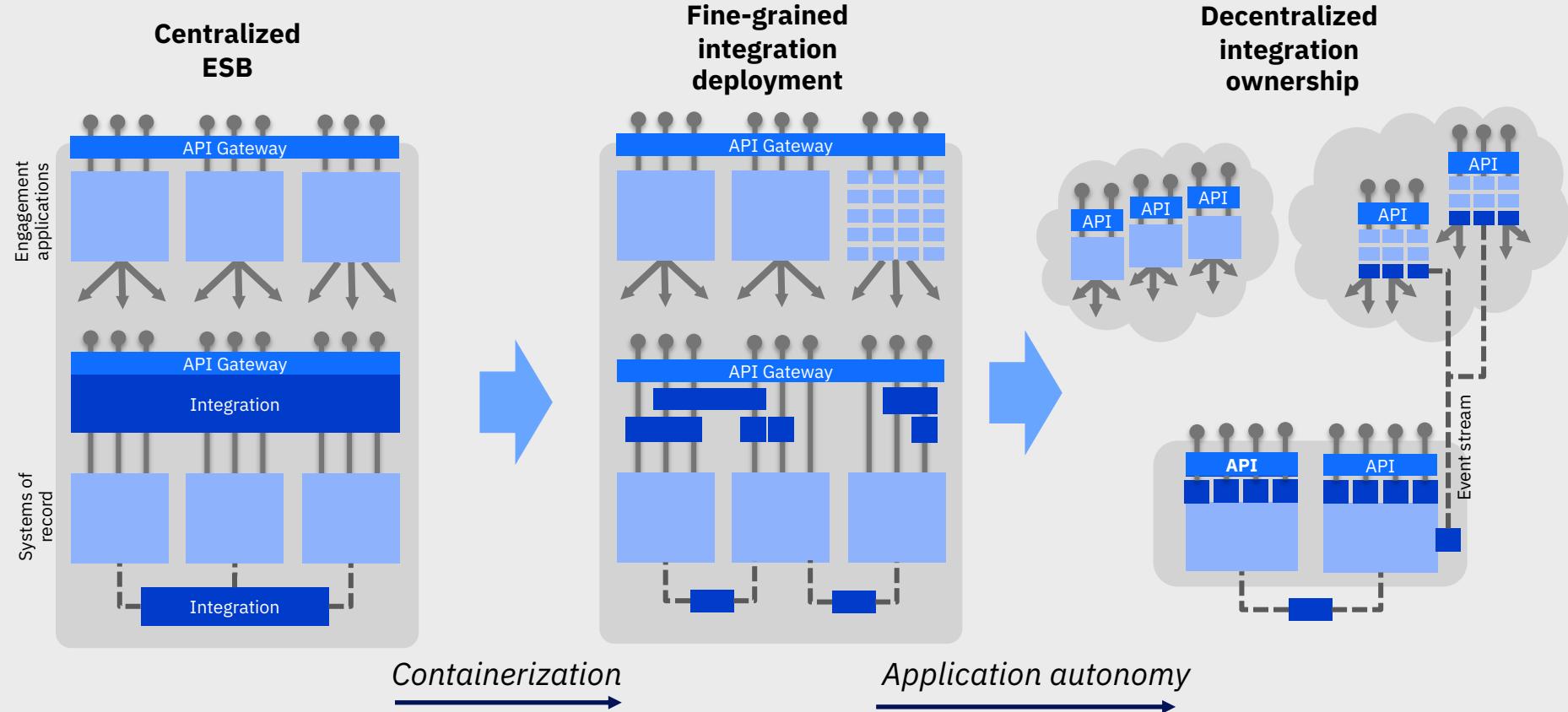
Use cases benefiting from elastic scaling

- Typical use cases in production
 - Rarely used functions
 - Functions with high workload variation
 - Elastic parallelization of batch
- Use cases beyond production
 - Prototyping
 - Performance testing

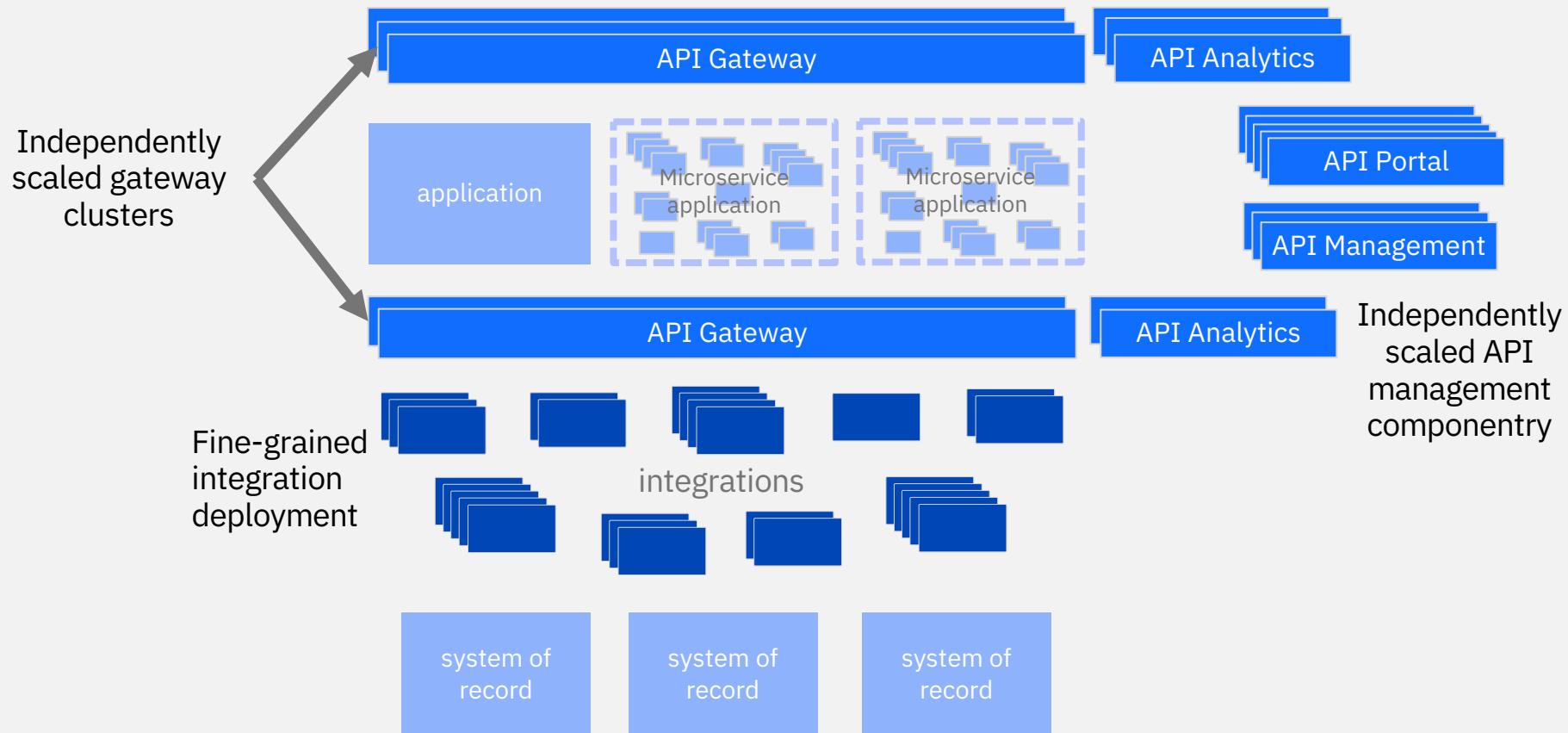


Almost all use cases have variable load at the individual function level

Evolution to agile integration architecture – detail view



Capabilities scale in different ways



Scaling across environments

AppDev

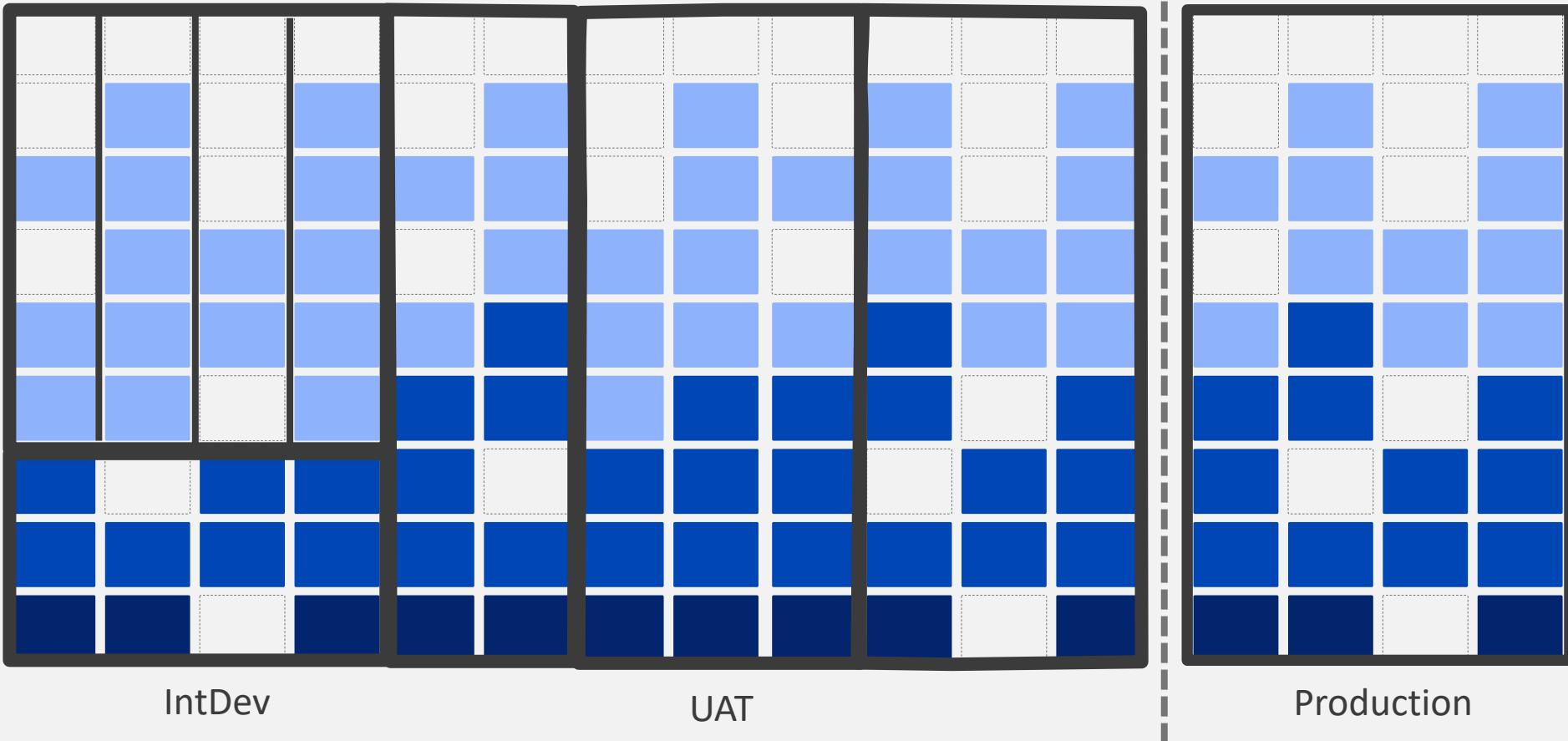
SysTest

Performance

IntDev

UAT

Production



Scaling across environments

AppDev

SysTest

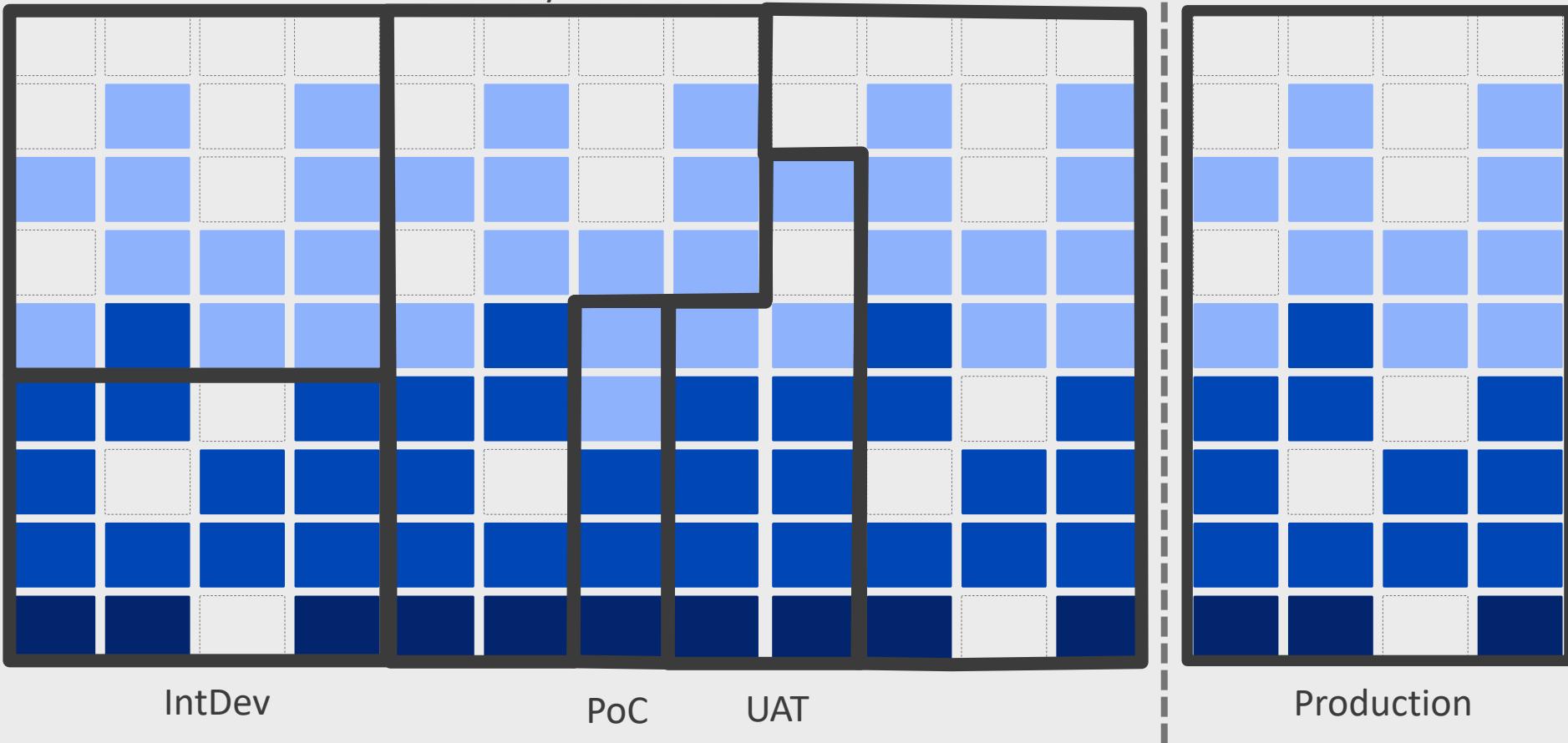
Performance

IntDev

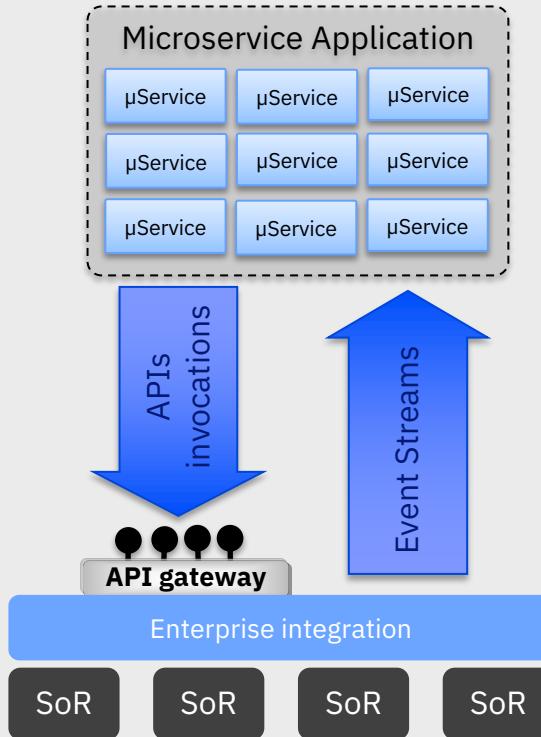
PoC

UAT

Production



Creating truly independent digital applications requires asynchronous communications as well as APIs



Truly independent, decoupled microservice components enable

- **Agility:** Innovate rapidly without affecting other components
- **Scalability:** Scale only what you need, and only when you need to
- **Resilience:** Fail fast, return fast, without affecting other components

To provide those benefits they need to be independent of the systems of record

- **APIs:** Are simplest to use, but create a real-time dependency
- **Event streams:** *Enable microservices to build decoupled views of the data*

Caching – but in which layer?

Pros

Reduces load on all other layers.
App can potentially work offline
Makes app extremely responsive

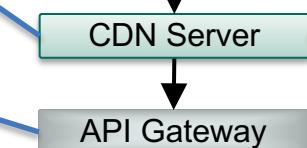
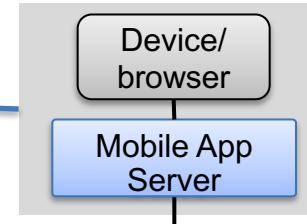
Reduces load on API Gateway and all layers below.
Closest geographical point-of-presence
Uses existing internet capability (via HTTP headers)

Reduces load on layers within enterprise.
API specific caching independent of application.
Cache consistent with API granularity

Reduces load on layers from application down.
Enables state free scalability for reference data
Writable cache options (with caution)
Compositions can benefit from fine grained caching.

Reduces load on database
Writable cache options with deep locking possibilities
Cache with understanding of the application
Application native data model can be used
Data relationships within cache are acceptable
Easiest point for accurate cache invalidation.
Further scale with grid compute
Preload closer to data store data model

No amount of caching at other levels is a substitute for a well designed, organised and tuned database. Modern databases (e.g. NoSQL) need attention too.



Cons

Can't share cache across users
Cache invalidation can be very challenging
Do you own the device app or have any controller over its design?

Read cache only.
Should you terminate HTTPS at the CDN?
Is asynchronous cache purge sufficient?
What cache visibility do you have?
Will you get re-use across regions?
How will you test its effectiveness?

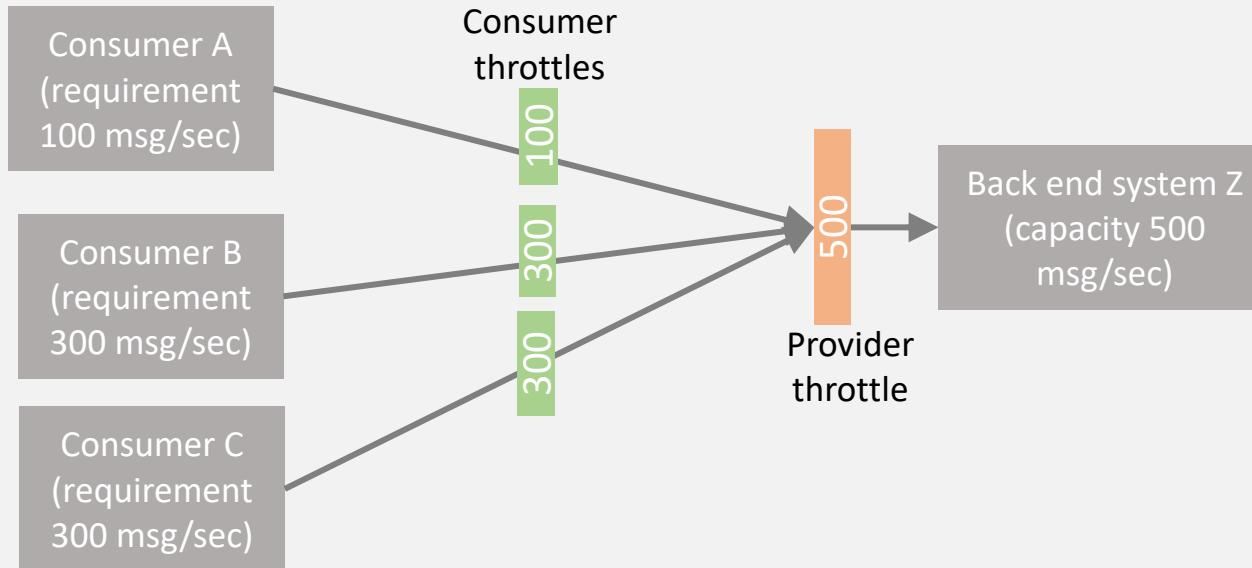
Must terminate HTTPS for full benefit.
Read cache primarily
How is cache invalidation performed?

Writable cache patterns can interfere with application design
Cache invalidation may require application knowledge.

Adds complexity to application build
Data model often different to API, so translation at other layers.
Change the application anyway or is it fixed?
What's the application code change cycle?

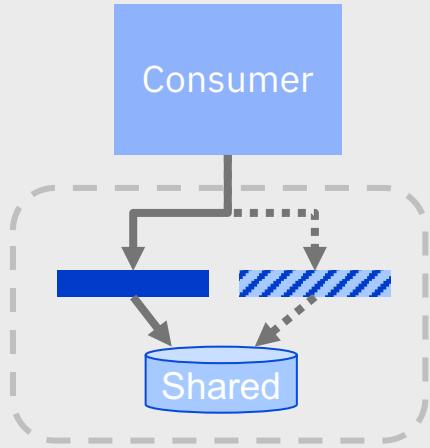
No reduction in load on application or layers above.
Database is the furthest distance from the client.
Do you have access to adjust the database?
Can you be sure you won't destabilise the application?

Managing throughput effectively requires multiple throttling points



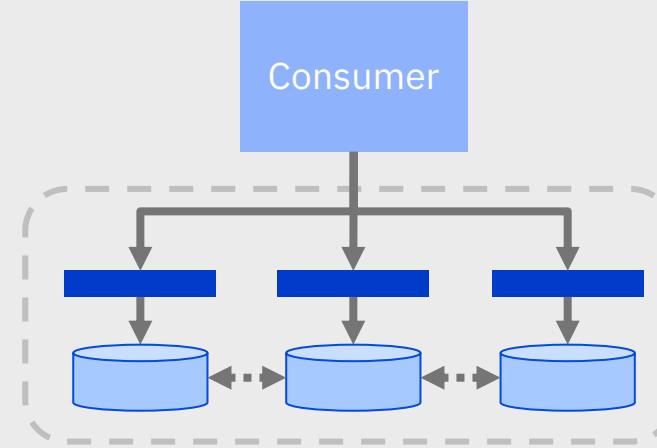
- No throttling – back end system easily overloaded.
- Provider only throttling – Back end system protected, but consumers can still take more than they are contracted for.
- Consumer only throttling – Enables prioritisation of consumers. Aggregate of consumers must be limited to back end capacity or risk overloading.
- Consumer and provider throttling – Consumers forced to behave to SLA, and back end system protected.

Simplified scaling and high availability



Active/passive with shared data

One master available at a time
Failover requires detection and warm up
Dependent on shared persistence



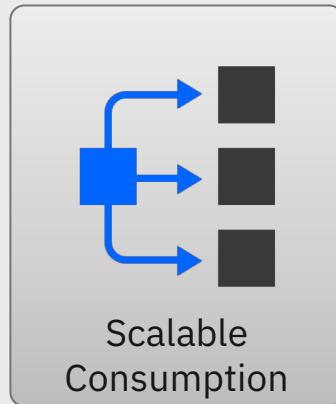
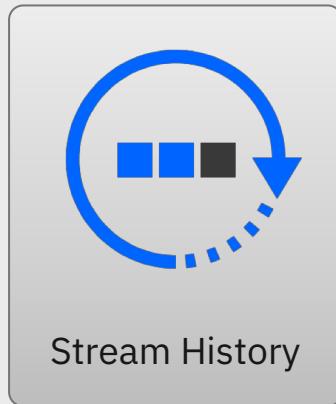
Active/active with duplicated data

Horizontal scaling

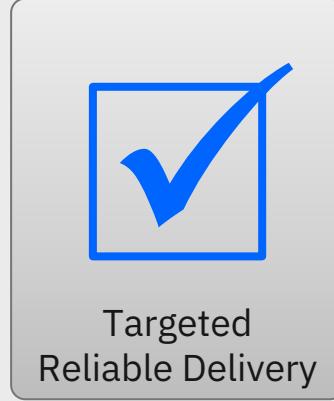
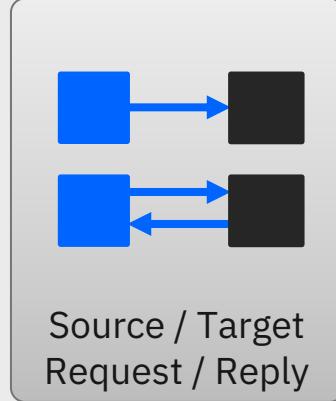
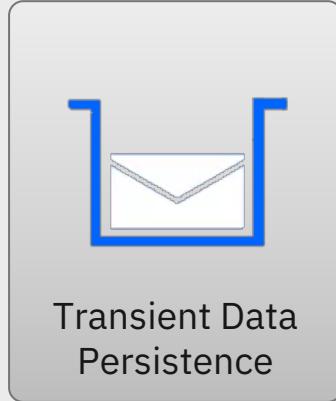
Instant failover

Replicated local persistence

Events (notifications)

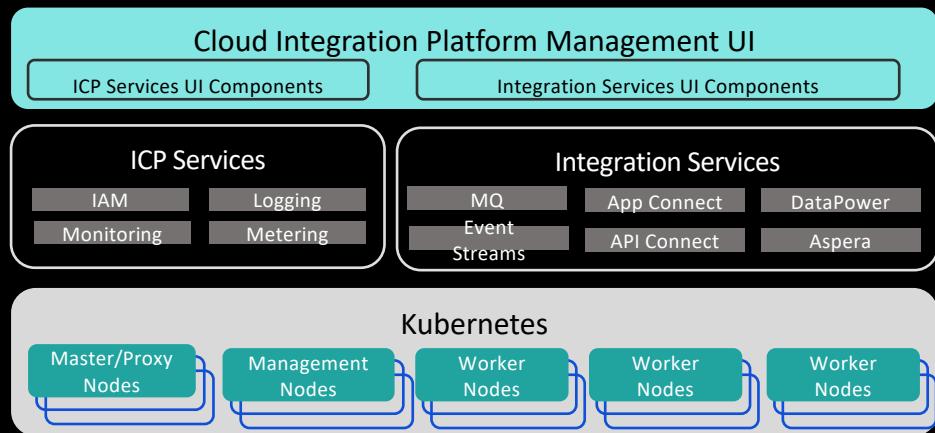


Messaging (commands)



Cloud Integration Platform Architecture

- Cloud Integration Platform (CIP) deploys and manages instances of **Integration Services** running on a Kubernetes Infrastructure
- Instances of Integration services are deployed individually as needed to satisfy each use-case.
- Services can be deployed in **Highly Available** topologies across multiple Kubernetes worker nodes or non HA on a single worker.



Scalability for Integration services.



Integration Services run on the **worker nodes**.

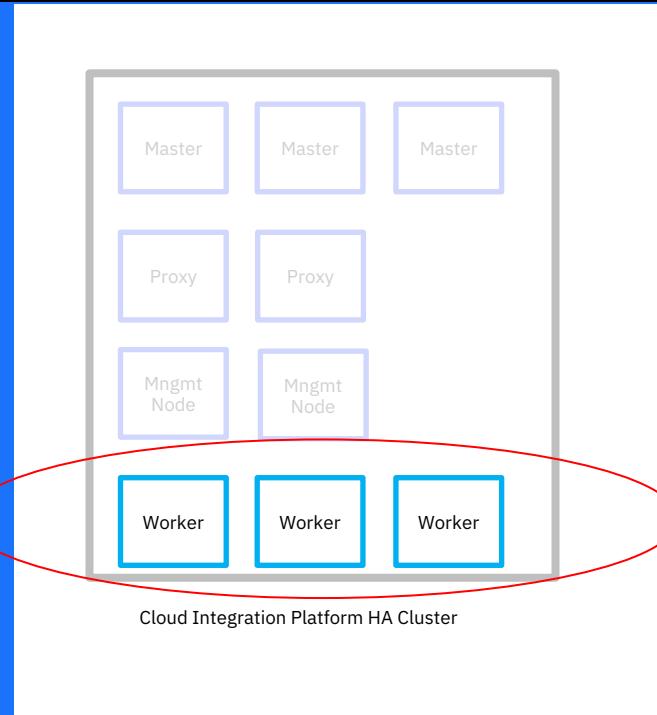
The Cloud Integration **SolutionPak** is composed from the **CloudPaks** making from the component products.

IBM cloudpaks are developed by the product development teams to embody best practice for deploying the product onto kubernetes as a secure, **scalable** Highly Available deployment.

Typically, all you need to do is scale the number of instances of each service and the CPU and memory it is allotted.

Kubernetes takes care of:

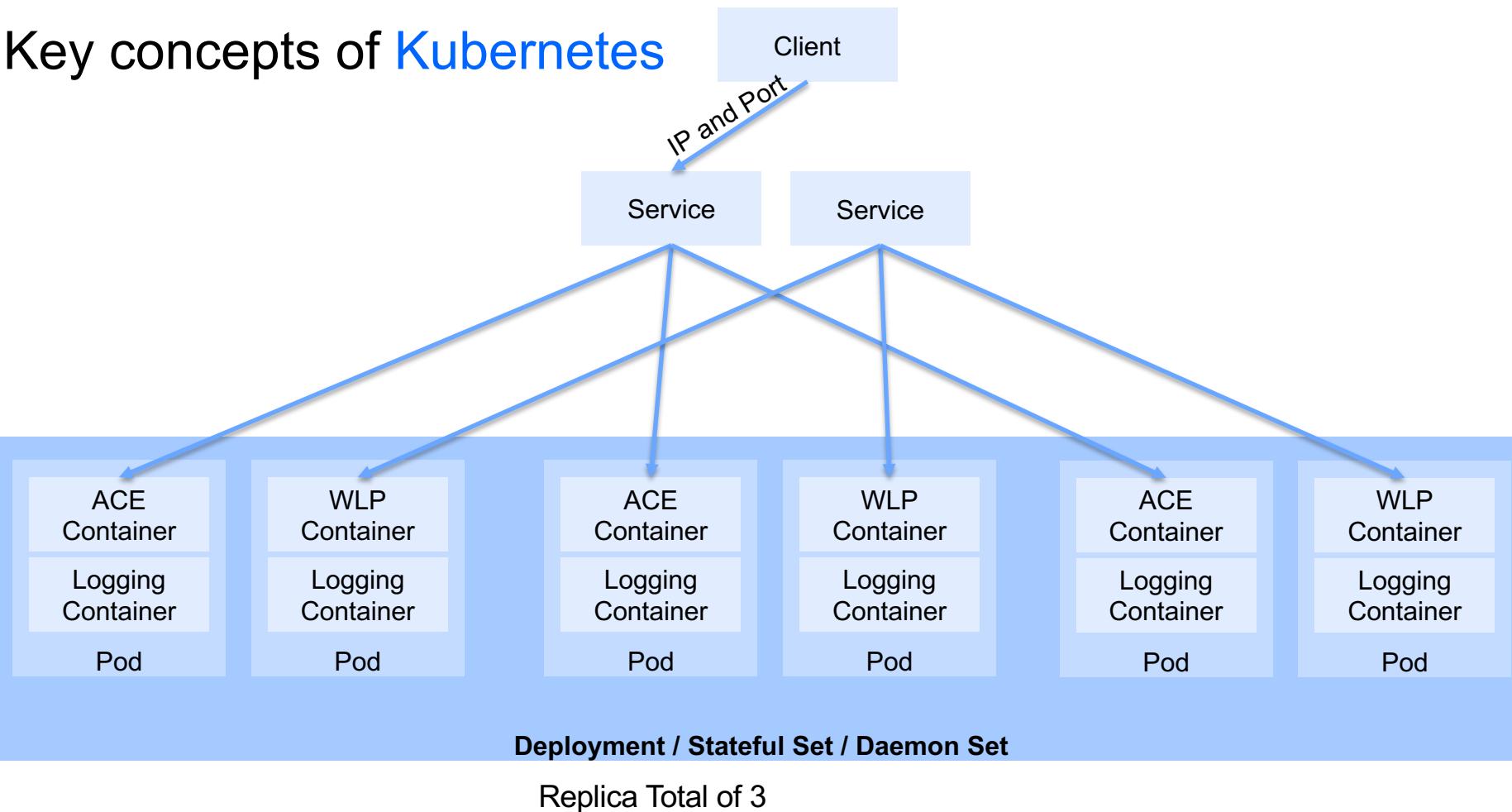
- Running appropriate numbers of instances of the solutionpak, spread across the available workers.
- Mixing together the workloads on the available worker nodes taking account the resources they need.
- Restarting workloads that fail and recovering from failed nodes by scheduling workloads onto alternative nodes.



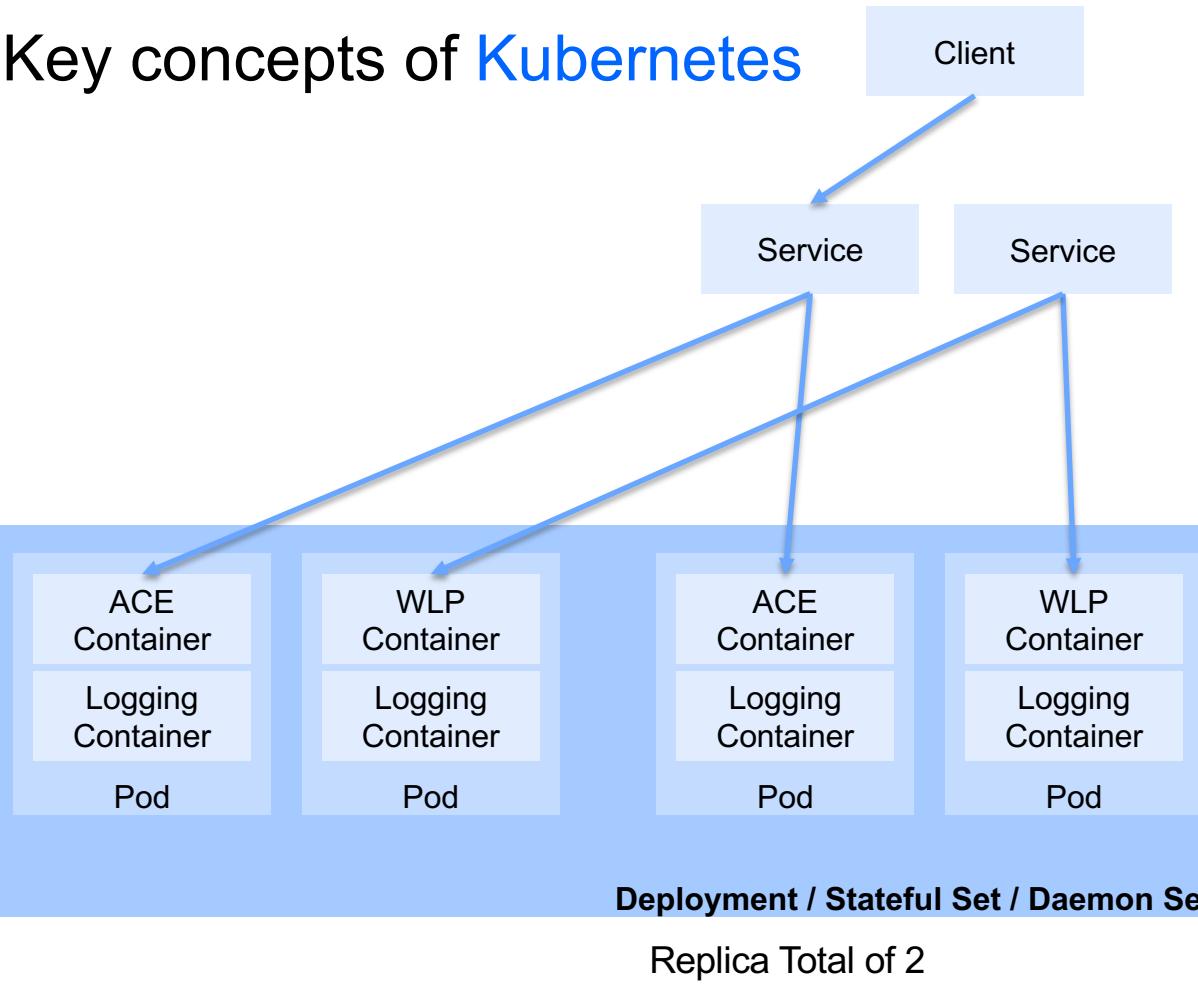
* Some of the component products require 2 worker nodes for active/standby style HA and others require 3 or more workers for quorum style deployment.

** In larger clusters the nodes should be spread between 3 or more availability zones.

Key concepts of Kubernetes



Key concepts of Kubernetes



Scaling options



- **Horizontal scaling** – change the number of instances
 - Typical scaling approach used in Kubernetes.
- **Vertical Scaling** – Change CPU / Memory limits.
 - Requests
 - Requests.cpu
 - Requests.memory
 - Used at scheduling time. Pods are scheduled to nodes that have ‘unrequested’ resource.
 - Pod will not be scheduled to a node that is ‘full’
 - Limits
 - Limits.cpu
 - Limits.memory
 - CPU– affects CPU allocated to pod **when node is busy**
 - Memory – Pods that exceed their memory limit likely to be terminated

Setting scale at deployment time



- Each of the helm charts includes replica count, CPU and memory limits for their subsystems.

Replica count i

Resources for ACE (without MQ) deployments

Resources settings for the running pods

CPU request * i

Memory request * i

CPU limit * i

Memory limit * i

Scaling manually after deployment

- Deployments and replica sets can be scaled manually.
- Kubernetes creates new pods and schedules them where resources are available.

The screenshot shows the IBM Cloud Private interface for managing deployments. At the top, there's a navigation bar with 'IBM Cloud Private' and links for 'Create resource', 'Catalog', 'Docs', and 'Support'. Below the navigation is a search bar and a 'Create Deployment' button. The main area is titled 'Deployments' and shows a table of current deployments across different namespaces. One row is highlighted, showing a deployment named 'ace-nomq-ibm-ace-server-icp-prod' in the 'mq' namespace with 3 desired, 3 current, and 3 ready pods. A context menu is open over this row, with options 'Edit', 'Scale', and 'Remove'. A modal window titled 'Scale Deployment' is displayed, asking 'How many instances do you want to scale to?' with a dropdown set to '4'. At the bottom of the modal are 'Cancel' and 'Scale Deployment' buttons. The overall interface has a clean, modern design with a light blue header and white background.

Name	Namespace	Desired	Current	Ready	Available	Created
ace-nomq-ibm-ace-server-icp-prod	mq	3	3	3	3	1 hour ago
r09aaff73f9-ui	apic	1	1	1	1	
r09aaff73f9-lur-v2	apic	1	1	1	1	
r09aaff73f9-ldap	apic	1	1	1	1	

DEPLOYMENT
Scale Deployment
How many instances do you want to scale to?
Instances:
4
Cancel Scale Deployment

Name	Namespace	Desired	Current	Ready	Available	Created
ace-nomq-ibm-ace-server-icp-prod	mq	4	4	4	4	1 hour ago

Scaling automatically based on CPU Load.



- Kubernetes scaling policies can be set based on CPU Load.
- Watch CPU load on pods and scale services up and down between defined min and max.
- Today this is based on CPU Load.
- ICP has experimental capability to base this on other metrics from the monitoring service/prometheus

Reference:

<https://developer.ibm.com/integration/blog/2018/07/27/customizing-docker-images-helm-charts-iibace-deployment-icp-assigning-scaling-policies-pods/>

The screenshot shows the 'Create Policy' dialog box. The 'Name' field contains 'ace-custom-scaling-policy'. The 'Namespace' dropdown is set to 'default'. The 'Scale target' dropdown is set to 'ace-custom-ibm'. Under 'Minimum replications', the value is '1'. Under 'Maximum replications', the value is '3'. In the 'Target CPU' input field, the value '10' is entered. Below the dialog, the 'Events' tab is selected in the navigation bar. A table displays three events: 1. A 'Normal' event from a 'deployment-controller' source with a count of 1, reason 'ScalingReplicaSet', and message 'Scaled up replica set ace-webuicustom-ibm-ace-6f6b67f4f4 to 1'. 2. A 'Normal' event from a 'deployment-controller' source with a count of 1, reason 'ScalingReplicaSet', and message 'Scaled up replica set ace-webuicustom-ibm-ace-6f6b67f4f4 to 3'. 3. A 'Normal' event from a 'deployment-controller' source with a count of 1, reason 'ScalingReplicaSet', and message 'Scaled down replica set ace-webuicustom-ibm-ace-6f6b67f4f4 to 1'. The second and third messages are highlighted with red boxes and circled.

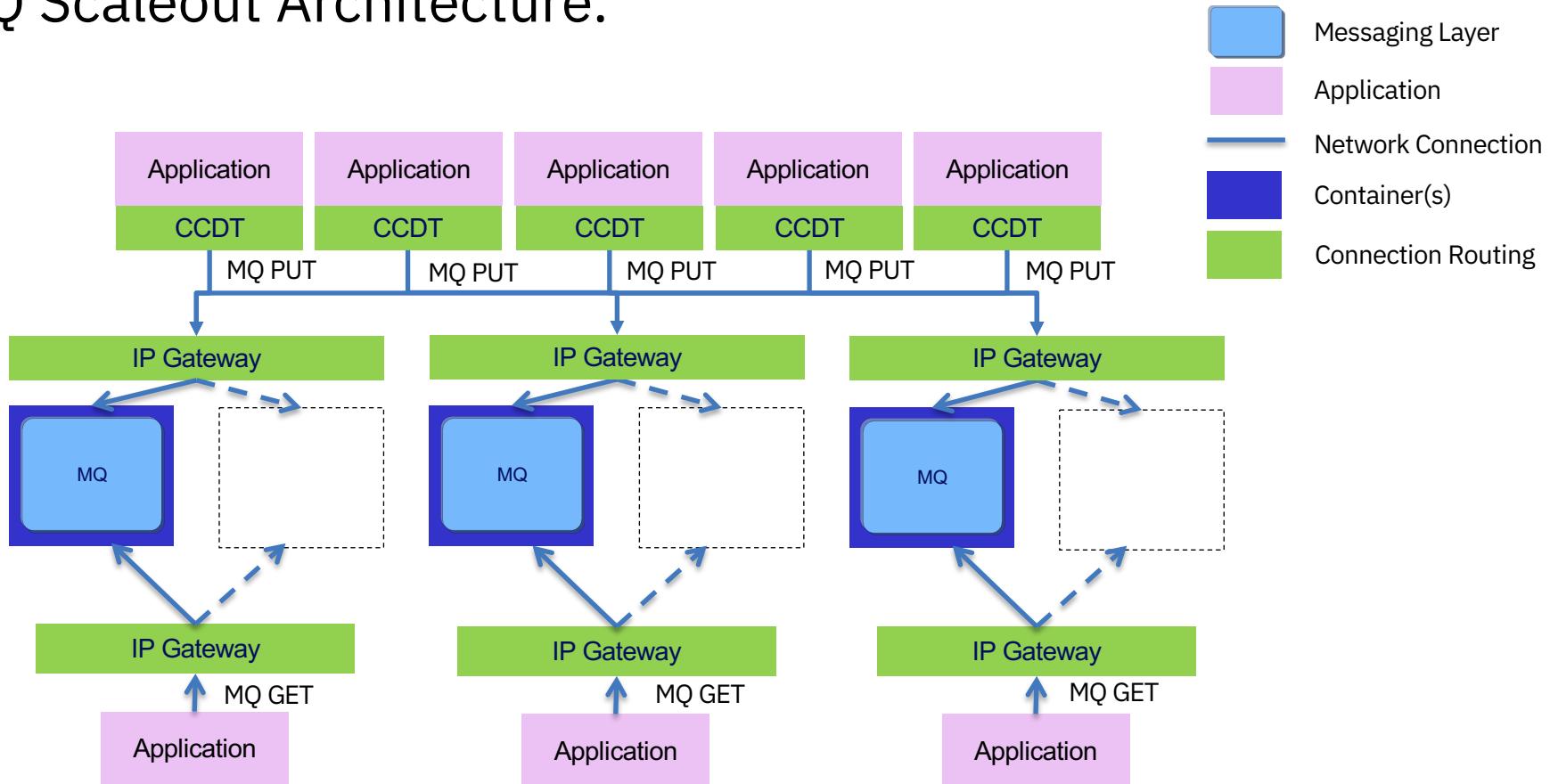
TYPE	SOURCE	COUNT	REASON	MESSAGE
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set ace-webuicustom-ibm-ace-6f6b67f4f4 to 1
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set ace-webuicustom-ibm-ace-6f6b67f4f4 to 3
Normal	deployment-controller	1	ScalingReplicaSet	Scaled down replica set ace-webuicustom-ibm-ace-6f6b67f4f4 to 1

API Connect scaling policies.

- API connect creates scaling policies as part of a standard deployment.
- These can be adjusted to suit the organization needs.

Scaling Policies						All namespaces
Name	Namespace	Reference	CPU%	Min Replicas	Max Replicas	Create Policy
r09aaff73f9-a7s-proxy	apic	r09aaff73f9-a7s-proxy	70	1	1	Create Policy
r09aaff73f9-apim	apic	r09aaff73f9-apim-v2	70	1	1	Create Policy
r09aaff73f9-juhu	apic	r09aaff73f9-juhu	70	1	1	Create Policy
r09aaff73f9-ldap	apic	r09aaff73f9-ldap	70	1	1	Create Policy
r09aaff73f9-lur	apic	r09aaff73f9-lur-v2	70	1	1	Create Policy
r09aaff73f9-ui	apic	r09aaff73f9-ui	70	1	1	Create Policy

MQ Scaleout Architecture.



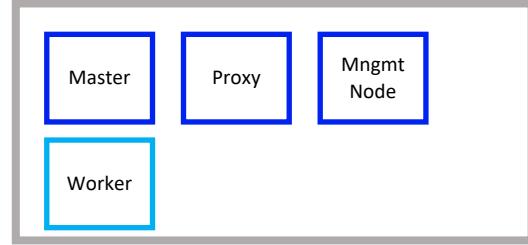
Container Platform Scalability



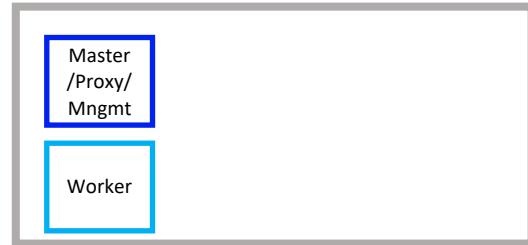
- Cloud Integration Platform is based on ICP Foundation.
- ICP Foundation is functionally identical to ICP Cloud Native edition. The only difference is licensing.
- All HA, scaling sizing and deployment information provided for ICP Cloud Native edition applies to Cloud Integration Platform.
- This presentation provides a summary. For more detail consult ICP Cloud Native edition documentation, articles and education. See the links page at the end of this presentation.

Scaling Kubernetes

- A kubernetes **Node** is a VM or bare metal machine which is part of a Kubernetes cluster.
- **Worker Nodes** run Integration Services. Enough CPU and memory are required to run all the scaled out Pods in your workload.
- **Master Nodes** run the services that control the cluster including the *etcd* database that stores the current state of the cluster.
- The load on the master nodes increases with the number of Nodes, Pods and other objects deployed. Master nodes must be scaled accordingly.
 - For very large clusters the *etcd* responsibility can be offloaded to separate *etcd* nodes
- **Proxy Nodes** transmit external requests to the services created inside your cluster. For large clusters use multiple proxy nodes and load balance across them with DNS or external load balancer.
- **Management Nodes** host management services such as monitoring, metering, and logging. As the number of worker nodes increases, management loads will scale proportionately.



Cloud Integration Platform
Cluster Node types



Cloud Integration Platform
Theoretical minimum cluster

Container Platform Scalability



	Node type	Number of nodes	CPU	Memory (GB)
Small Cluster	Master	3	16	32
	Management	2	8	16
	Proxy	2	4	16
	Worker	3+ (Max:20)	8	32
Medium Cluster	Master	3	16	32
	Management	2	16	32
	Proxy	3	4	16
	Worker	5+ (Max:70)	8	32
Large Cluster	Master node	3	16	128
	Management node	2	16	128
	Proxy node	3	4	16
	Worker node	500	8	32

Note that Master, management and proxy nodes are typically scaled UP before scaling out.

Example – CIP deployed in AWS

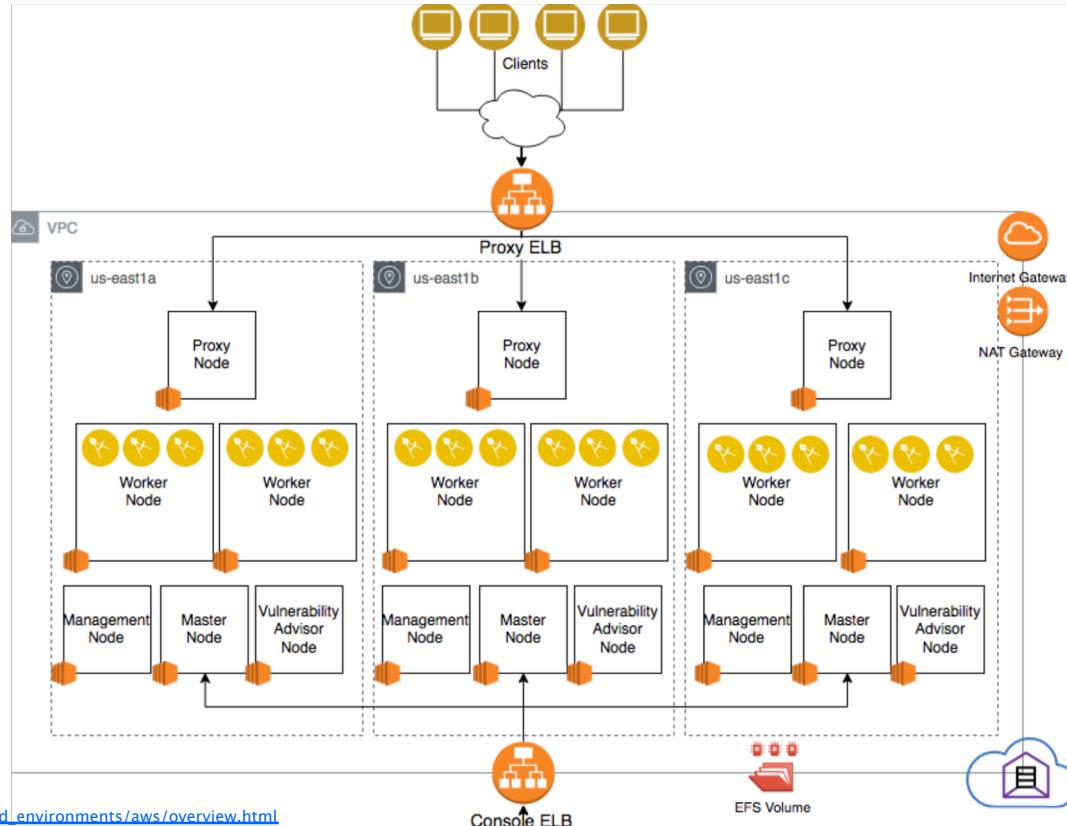
Leverage available zone

- Master/mgmt/va across available zone
- User application across available zone

AWS ALB/NLB

- Load balancer for management control plane
- Load balancer for user application
- Security group to control network access

EBS as persistent storage



Article: <http://ibm.biz/icponaws>

Documentation: https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/supported_environments/aws/overview.html

links to a quickstart: <https://aws.amazon.com/quickstart/architecture/ibm-cloud-private/>

Thank You

