Building MSA in Enterprise

Microservice adoption Pattern for Enterprise

Youngjoon Jeong, Sr Container Solutions Architect, AWS WWSO
Yongsuk Kwon, Principal Engineer, Samsung CP Solution Dev Group

Table of contents

aws

Why MSA matter in Enterprise

aws

New business situation for new enterprises

Business confidence and consumer sentiment are not seeing eye to eye…… yet.

By the End of 2021, 80% of Enterprises will shift to Cloud-Centric Infrastructure and Applications twice as fast as before the pandemic.

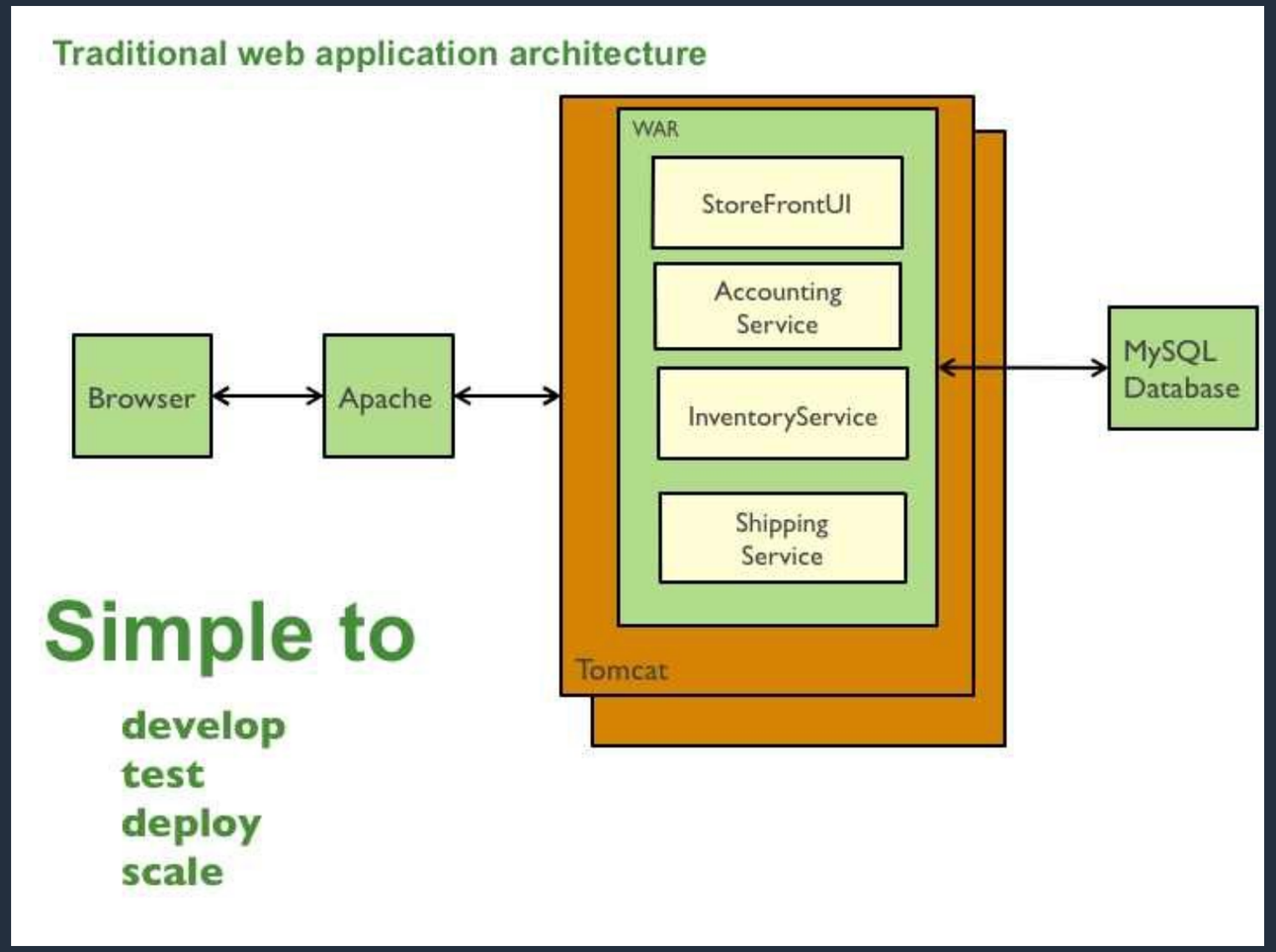
In 2022, Enterprises focused on modernisation will have adapted to disruption and respond to new conditions 50% faster.

Through 2022, coping with technical debt accumulated during the pandemic will shadow 70% of CIOs, causing financial stress and drag on IT agility

aws

Old School Operation & Automation

- Monolithic - Automation for monolithic applications
- Familiar - Existing organizational processes and engineers are familiar with
- Trusted - A lot of investment has already been made and it has high reliability because it is working well.



aws

Reality

- Legacy IT Strategies Aren’t Prepared for Change
- Legacy Systems Are Not Cost-Effective to Manage

aws

Core concept

MSA Concept & Why MSA matter in Enterprise

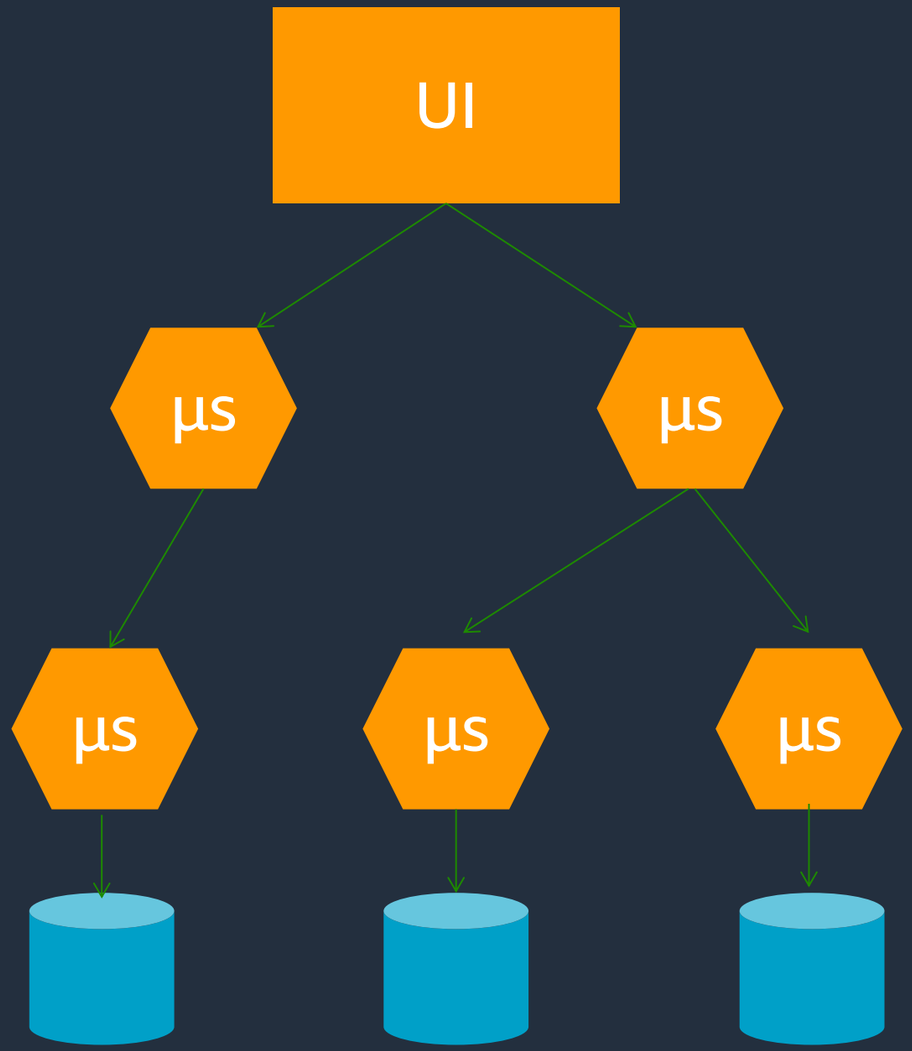
MSA Core Concept

1. Independently deployable
2. Loosely coupled
3. Highly maintainable and testable
4. Owned by a small team
5. Organized around business capabilities

1. Extension cube of MSA
2. Modularity
3. Data as Service

aws

MSA Core Concept



- Decomposition (Refactoring)
  - Anti-corruption layer
- Data Management
  - Database per Service, Shared Database
  - API Composition, CQRS, Domain Event, Event Sourcing
- Testing

aws

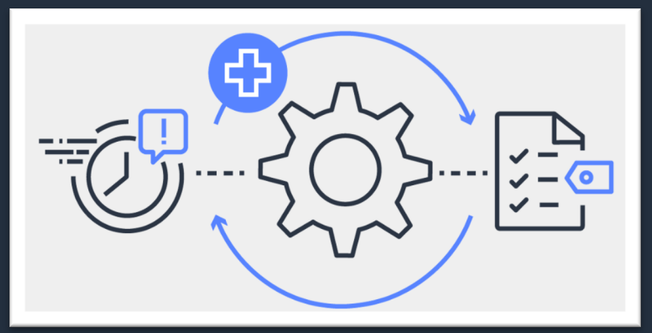
Base Rule for MSA in Enterprise

Infrastructure actively responds to all changes

⇒ Next Gen Infra is DevEverything

Security

Reliablity



Deploy

aws

Evolution of Operation & Infra

aws

Key point of Operation & Infra for MSA

Speed

Improve developer velocity with consistent environment

Agility

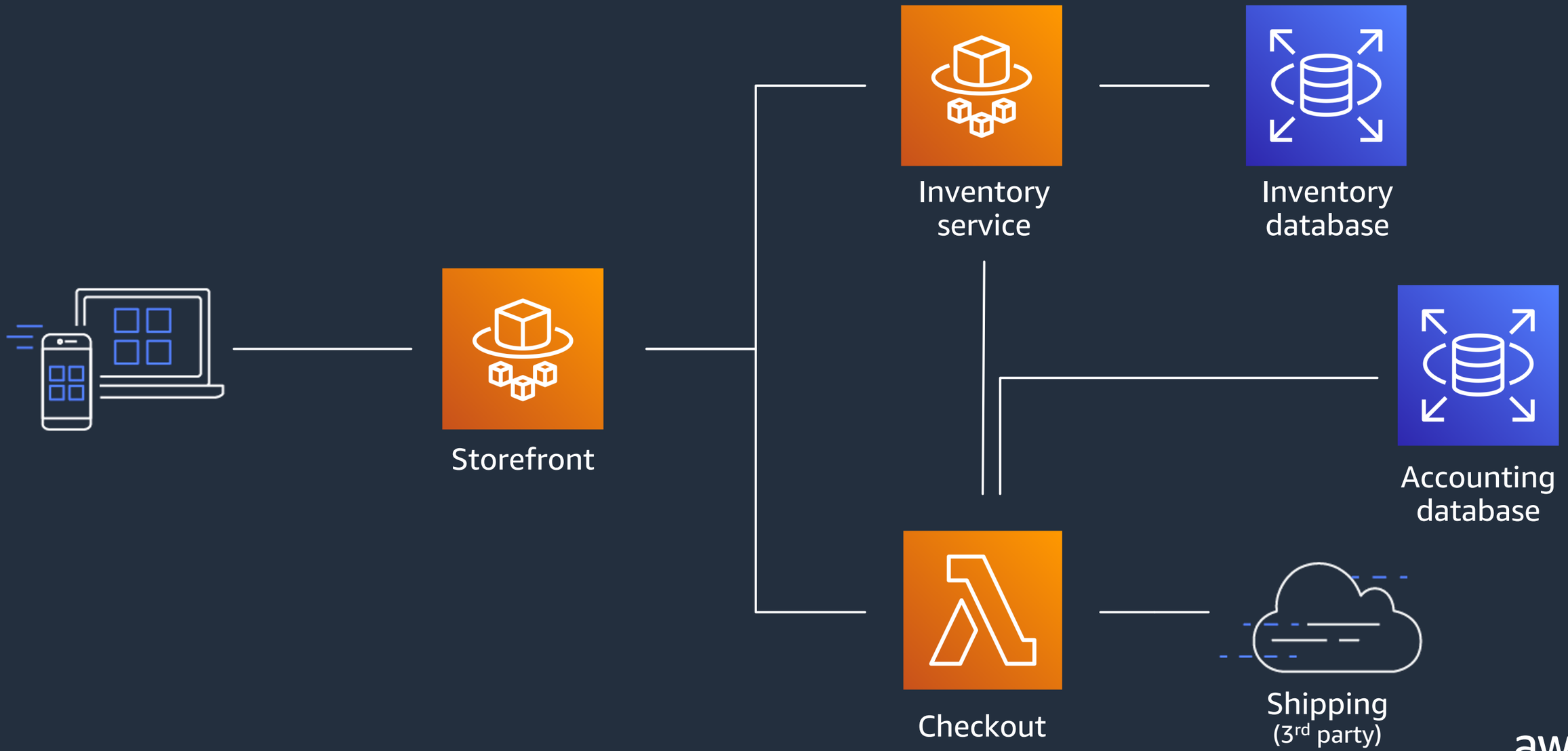
Automation increases ability to test and iterate

Security

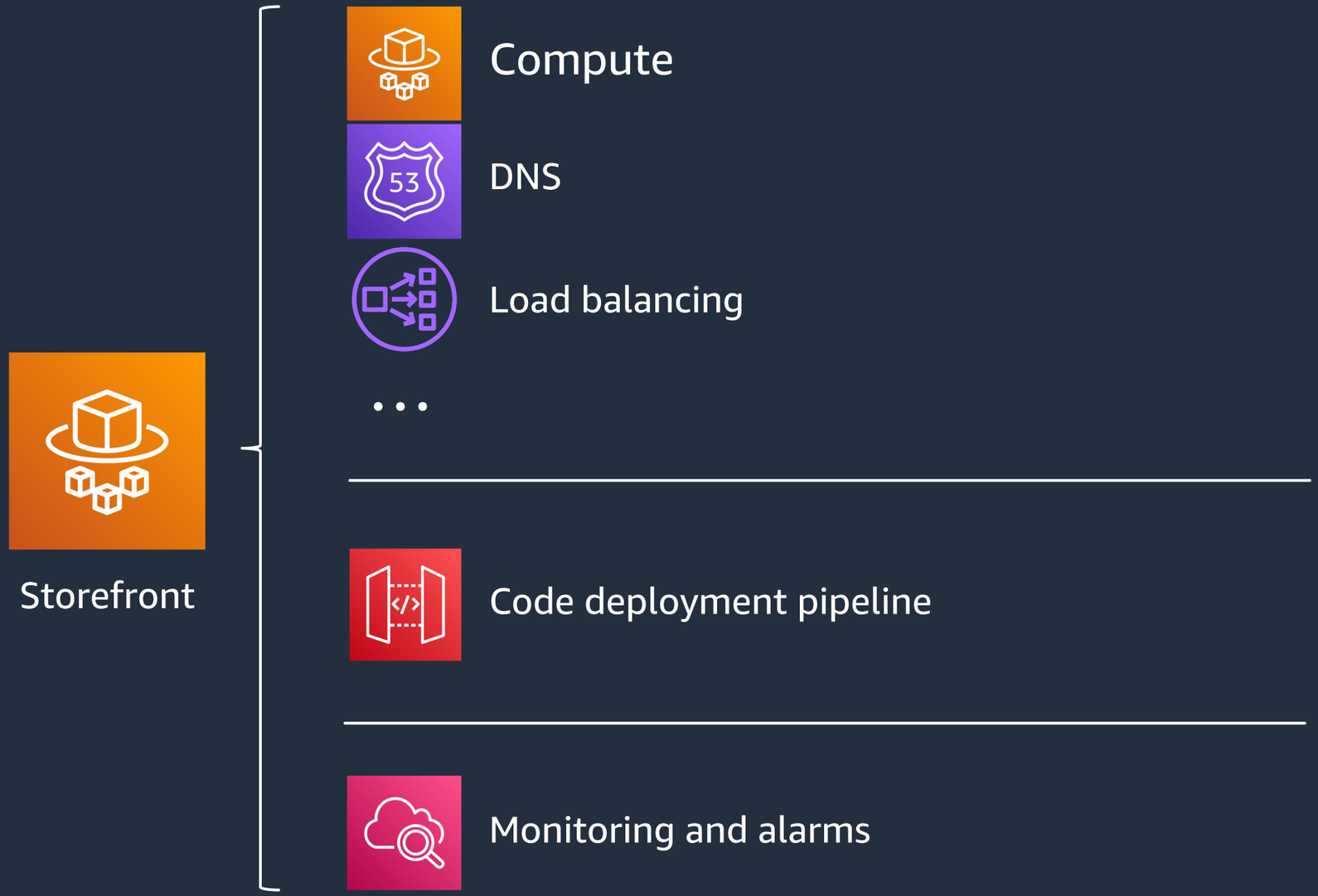
Integrated security tooling and governance controls

aws

Using microservices for speed and agility



“Simple” is not so simple



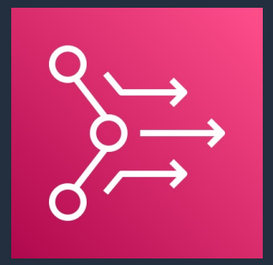
Application management as a spectrum

PaaS

Platform as a Service

- ↓ Flexibility & Control
- ↑ Developer Productivity

Ideal



- ↑ Flexibility & Control
- ↑ Developer Productivity

IaaS

Infrastructure as a Service

- ↑ Flexibility & Control
- ↓ Developer Productivity

Key point of Operation & Infra

- Security
    - Structure that can prevent data leakage as well as mistakes
    - Security for calls between services
    - Audit and response structure for worker node/network/storage options
- Reliability
    - Utilize infrastructure templates that can deploy services in a stable structure
    - Basic configuration for control plane/data plane/network structure
    - Backup – Manage by backing up your data as well as the state of your infrastructure
- Deploy
    - Declarative type – Minimizes the effort to maintain service by maintaining the specified setting details
    - Code base – Easy to change at any time as it is managed as a code

aws

Security

- Authorization (IAM and Token)
- Pod Security
- Multi-tenancy
- Detective Control
- Runtime Security
- Infrastructure Security

aws

Reliability

- Recommendations
- Monitor Control Plane Metrics
    - API Server
    - etcd
- Cluster Authentication
- Handling Cluster Upgrades
- Running large clusters
- Know limits and service quotas

aws

Deploy

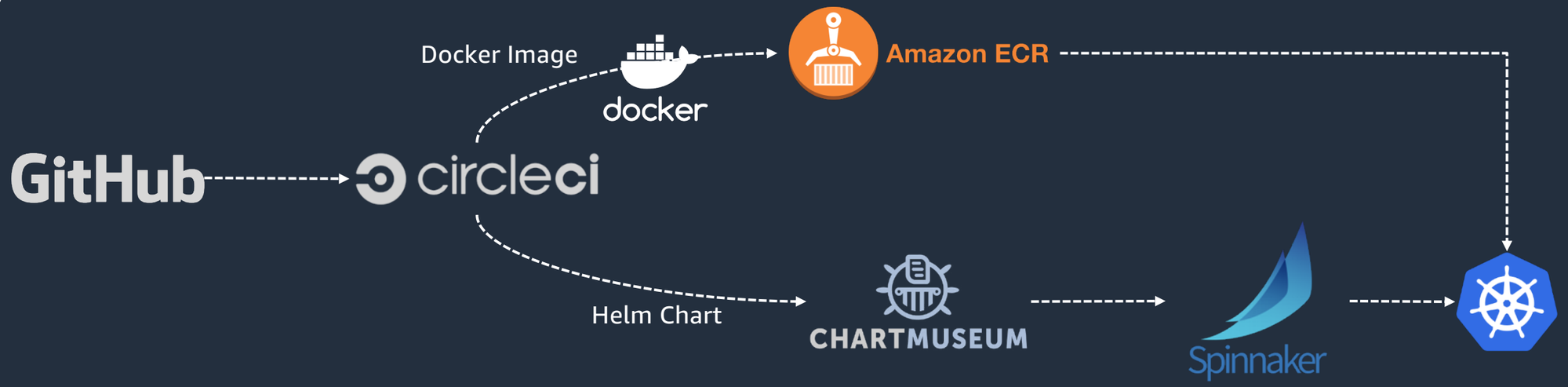
- CDK as Custom Framework
- Use Infra Template and Service Template
- Deploy Pipeline management
- Observation for Operating

https://github.com/superluminar-io/super-eks
https://github.com/aws-quickstart/quickstart-eks-cdk-python

aws

Deploy - Maintainability improve

Deployment



Infrastructure as a Code layer

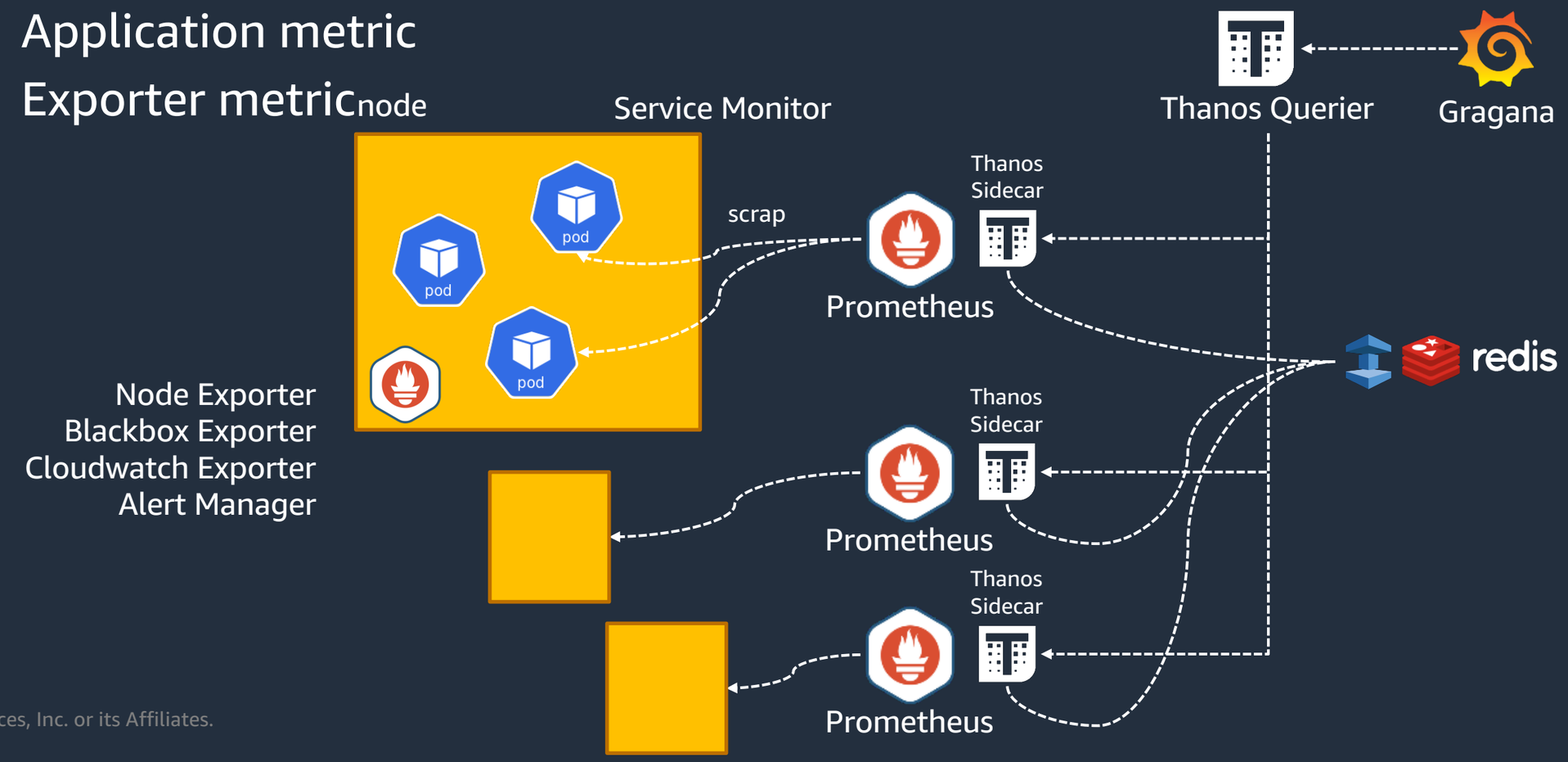
aws

Deploy - Observation for Operating

Monitoring

- Prometheus / Thanos
  - Application metric
  - Exporter metric

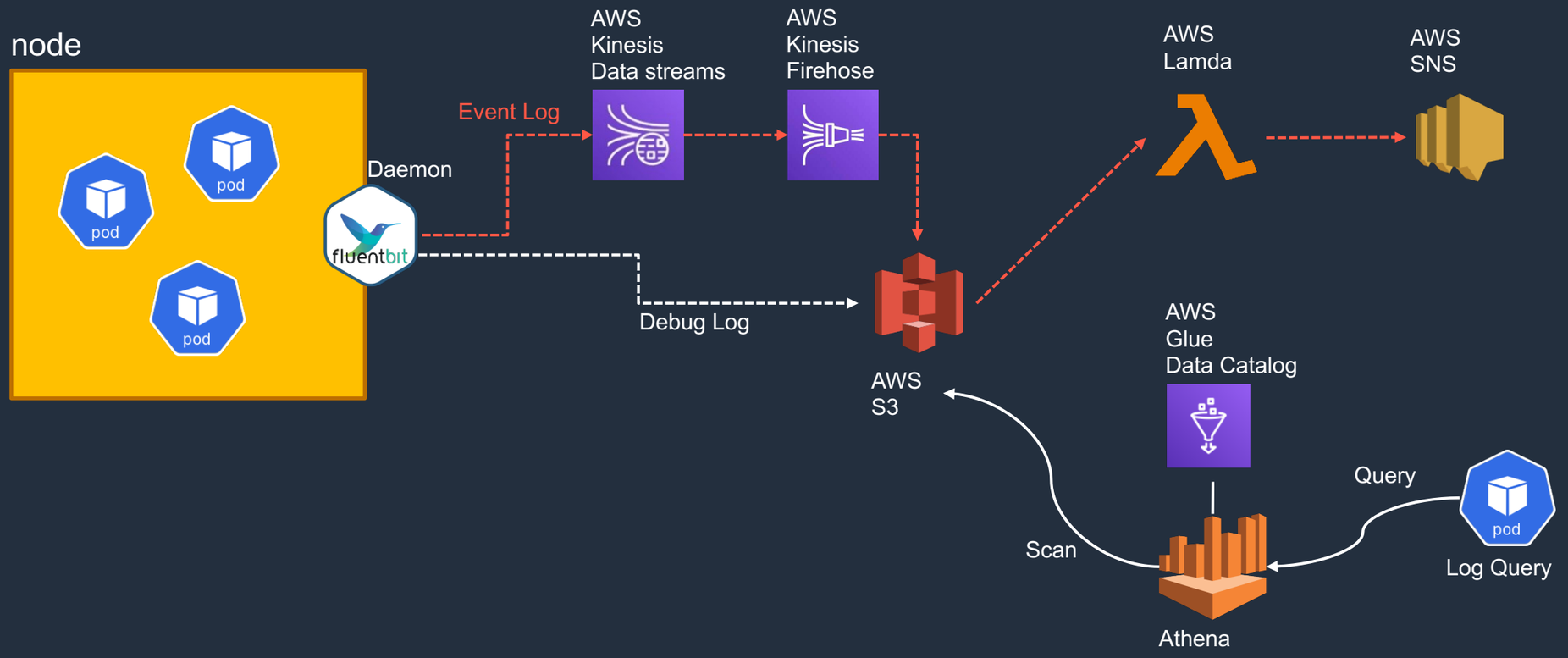
Metric Query Performance
- Thanos
- Histogram metric to SLA count bucket



aws

Deploy - Observation for Operating

Logging



Observability and Monitoring

Multi Cluster Monitoring

AWS Cloudwatch Insights interface offer this feature out of the box, so in a single interface you can monitor multiple clusters. Unfortunately, at the moment cross-account cluster monitoring is not supported.

At this moment the best way to monitor multiple cluster across multiple AWS accounts is by using Amazon Managed Prometheus service, the way how this work is that you can create a Managed Prometheus workspace in a single account, then having a Prometheus server on each EKS cluster and all of them forwarding metrics to the Managed Prometheus workspace

https://aws.amazon.com/blogs/opensource/setting-up-cross-account-ingestion-into-amazon-managed-service-for-prometheus/

Observability and Monitoring

Control Plane Monitoring

apiserver_request_total	Counter of apiserver requests broken out for each verb, dry run value, group, version, resource, scope, component, client, and HTTP response contentType and code.
apiserver_request_duration_seconds*	Response latency distribution in seconds for each verb, dry run value, group, version, resource, subresource, scope, and component.
etcd_request_duration_seconds*	Etcd request latency in seconds for each operation and object type.
etcd_object_counts	Number of stored objects at the time of last check split by kind

A high increase on total requests to the API server without an actual increase on the Data Plane workload means that some of the components used in the Kubernetes cluster is querying the API Server inappropriately and has to be further investigated.

aws

Observability and Monitoring

Control Plane Logging

It is not enabled by default as it does imply additional costs for Cloudwatch Logs storing and ingestion. Strictly speaking of troubleshooting the most important logs to detect problems in Kubernetes control plane are :

Kubernetes API server component logs (api) – Your cluster's API server is the control plane component that exposes the Kubernetes API. Useful to detect problems related to malformed requests, API server latency and timeout, Control plane unable to process request.

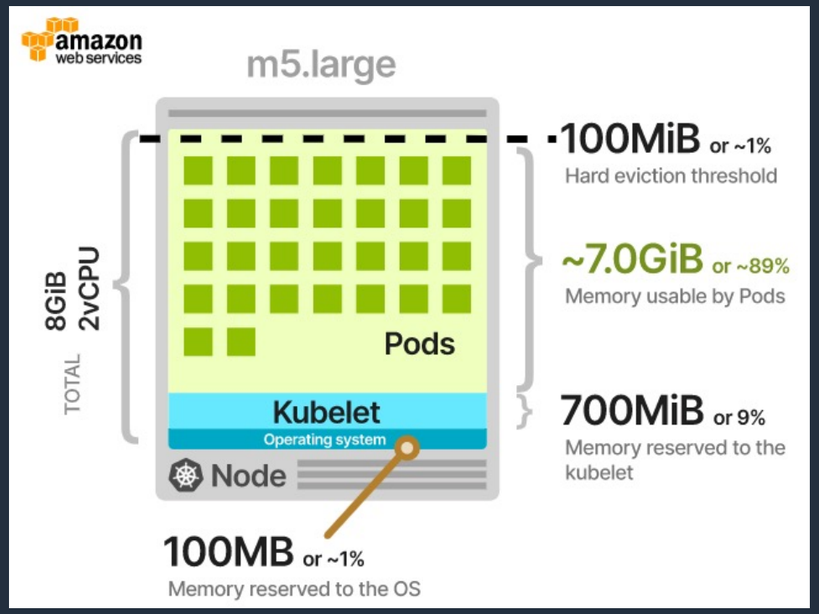
Audit (audit) – Kubernetes audit logs provide a record of the individual users, administrators, or system components that have affected your cluster. Useful to deep dive into components doing heavy usage of Kubernetes Control Plane.

Controller manager (controllerManager) – The controller manager manages the core control loops that are shipped with Kubernetes. Useful to detect reconciliation problems or failures on controllers execution.

Scheduler (scheduler) – The scheduler component manages when and where to run pods in your cluster. Useful to detect if Kubernetes scheduler isn’t capable to allocate pods in the Data Plane.

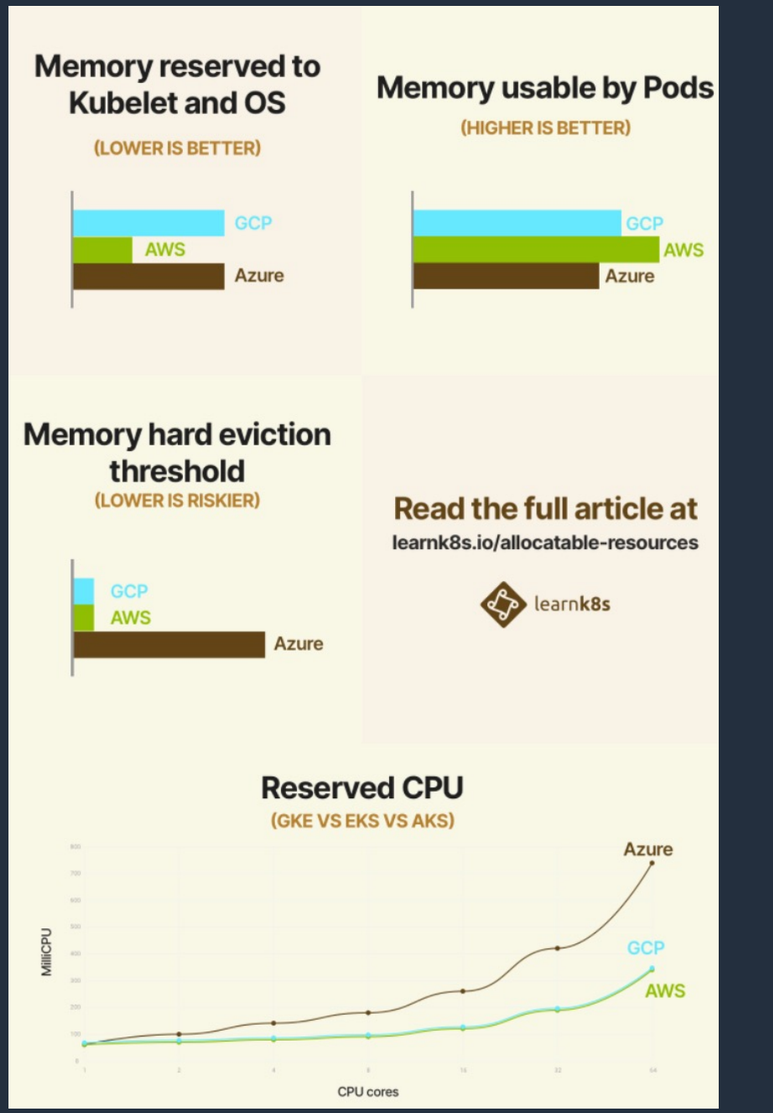
Cost Optimization

Allocating CPU & Memory



Reserved memory =
255MiB + 11MiB * MAX_POD_PER_INSTANCE

Reserved CPU =
6% of the first core
1% of the next core (up to 2 cores)
0.5% of the next 2 cores (up to 4 cores)
0.25% of any cores above 4 cores



An m5.large instance has 2 vCPU and 8GiB of memory:

574MiB of memory is reserved to the kubelet.
(255Mi + 11MiB * 29 = 574MiB)
An extra 100M of CPU and 100MB of memory is reserved to the Operating System
and 100MB for the eviction threshold.
The reserved allocation for the CPU is the same 70 millicores

https://learnk8s.io/allocatable-resources

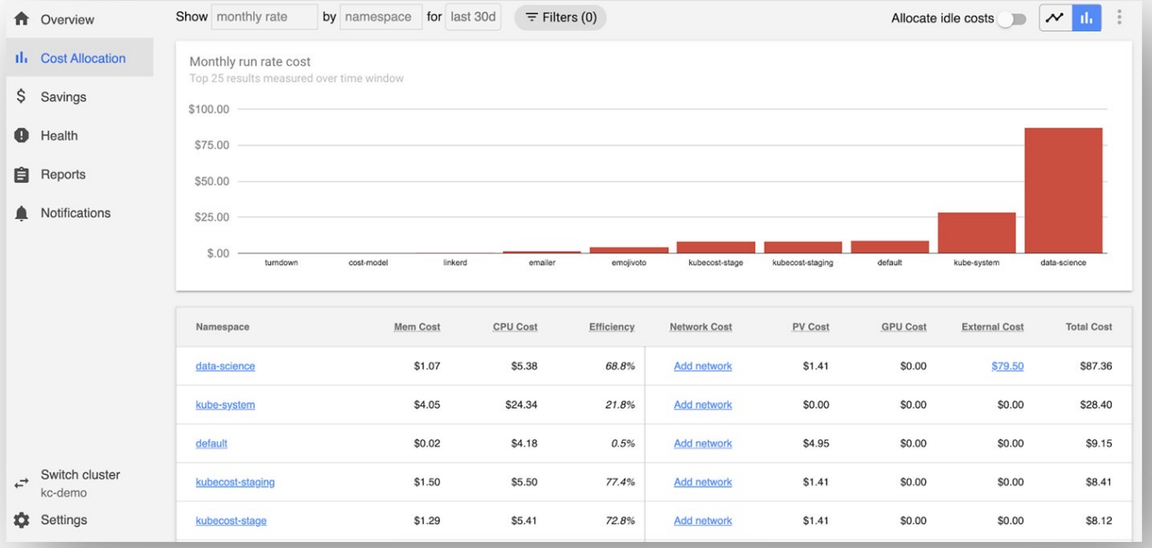
aws

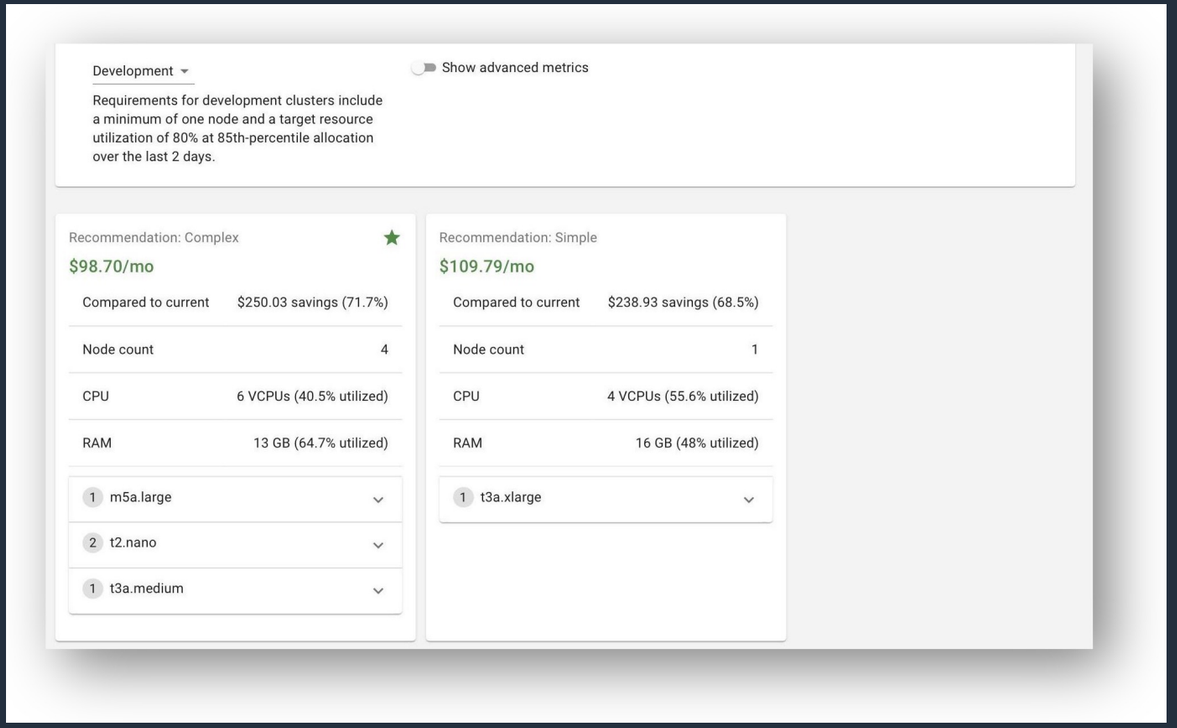
Cost Optimization

Kubecost

Kubecost, an open-core tool by Stackwatch, provides cost monitoring and capacity management solutions. One of its primary use cases is to give cost visibility across Kubernetes clusters. It uses three metrics to determine the cost of a workload:

1. Time in running state
2. Resources consumed or reserved
3. Price of resources consumed or reserved





https://aws.amazon.com/blogs/containers/how-to-track-costs-in-multi-tenant-amazon-eks-clusters-using-kubecost/

Security – Pod Security

- Restrict the containers that can run as privileged¶
- Do not run processes in containers as root
- Never run Docker in Docker or mount the socket in the container
- Restrict the use of hostPath or if hostPath is necessary restrict which prefixes can be used and configure the volume as read-only
- Set requests and limits for each container to avoid resource contention and DoS attacks
- Do not allow privileged escalation
- Configure your images with read-only root file system

aws

Security – Multi Tenancy

- Soft Multi-tenancy vs Hard Multi-tenancy
- Kubernetes Constructs
- Mitigating Controls

aws

Security - Detective Control

- Enable audit logs
- Utilize audit metadata
- Create alarms for suspicious events
- Analyze logs with Log Insights
- Audit your CloudTrail logs
- Use CloudTrail Insights to unearth suspicious activity
- Additional resources

aws

Security – Runtime

- Use a 3rd party solution for runtime defense
- Consider add/dropping Linux capabilities before writing seccomp policies
- See whether you can accomplish your aims by using Pod Security Policies (PSPs)

aws

Security – Infrastructure (Host)

- Use an OS optimized for running containers
- Treat your infrastructure as immutable and automate the replacement of your worker nodes
- Periodically run kube-bench to verify compliance with CIS benchmarks for Kubernetes
- Minimize access to worker nodes
- Deploy workers onto private subnets
- Run Amazon Inspector to assess hosts for exposure, vulnerabilities, and deviations from best practices

aws

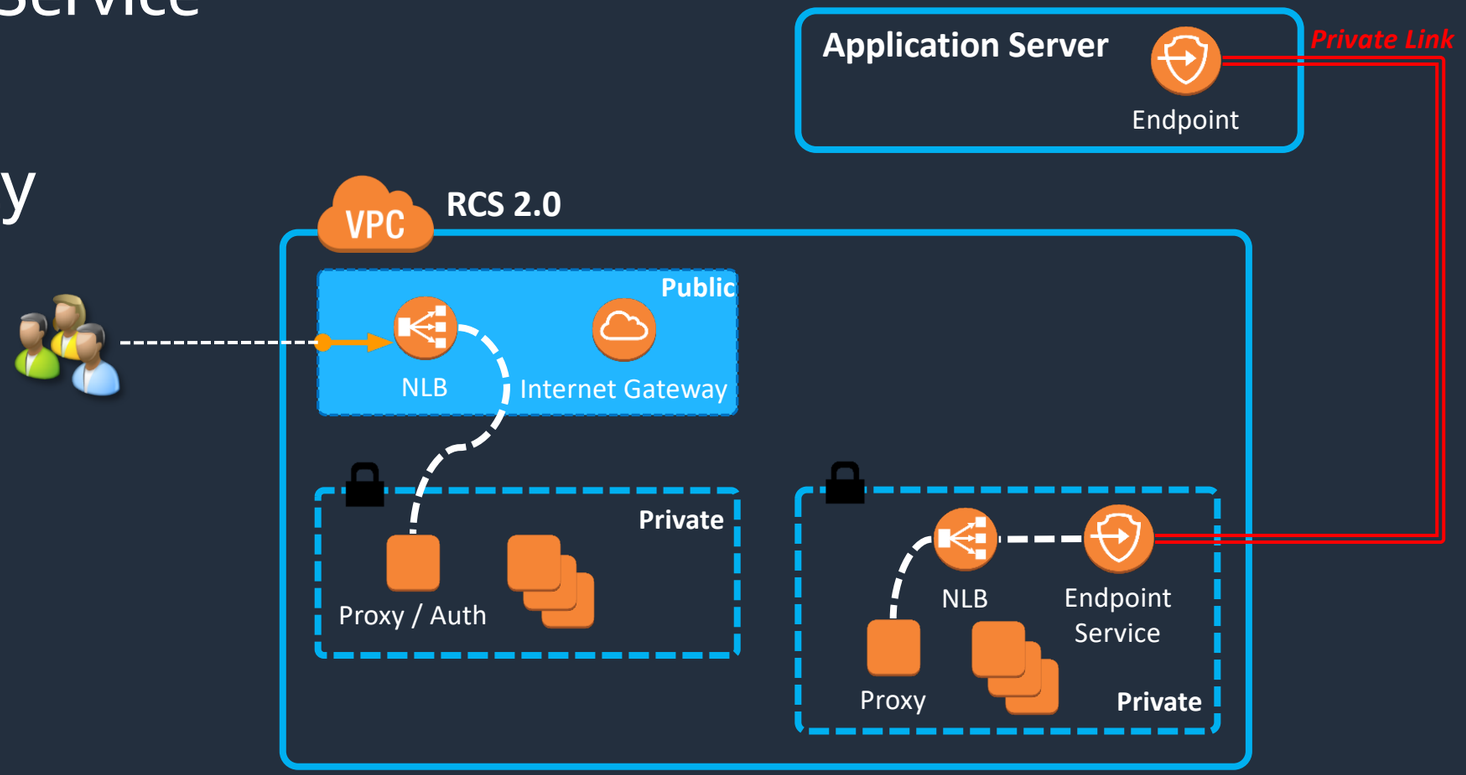
Security - Authorization (IAM and Token)

Private Link for S2S security

- Endpoint / Endpoint Service
- Private NLB

JWT token for P2S security

- Oauth2.0



aws

Reliability - Data Plane

- Use EC2 Auto Scaling Groups to create worker nodes
- Use Kubernetes Cluster Autoscaler to scale nodes
- Configure over-provisioning with Cluster Autoscaler
- Using Cluster Autoscaler with multiple Auto Scaling Groups
- Using Cluster Autoscaler with local storage
- Spread worker nodes and workload across multiple AZs
- Ensure capacity in each AZ when using EBS volumes
- Run node-problem-detector
- Reserving resources for system and Kubernetes daemons

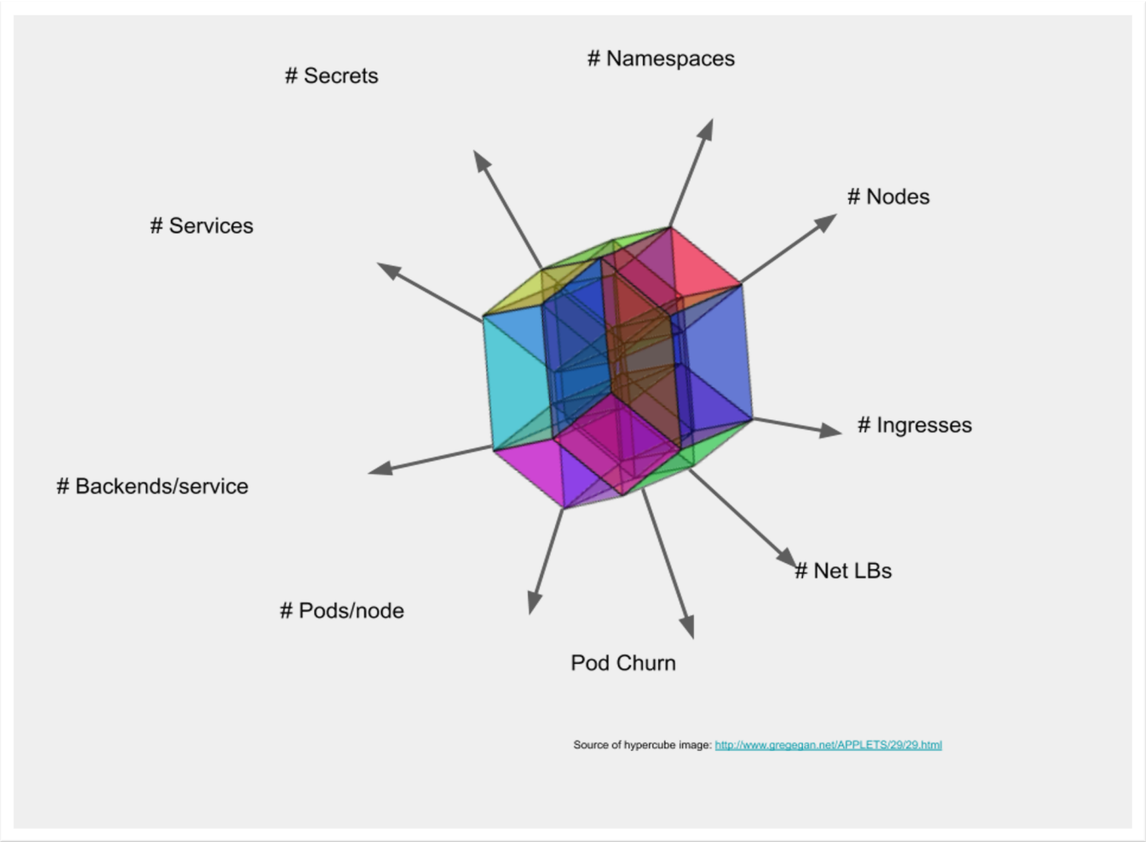
aws

Reliability - Network

- Deploy NAT Gateways in each Availability Zone
- Plan for growth
- Monitor IP address inventory
- Using public subnets for worker nodes
- Run worker nodes and pods in different subnets
- SNAT
- Size your subnets for growth
- Monitor CoreDNS metrics
- Use NodeLocal DNSCache
- Configure cluster-proportional-scaler for CoreDNS

aws

Reliability – Modular cube scale



Quantity	Threshold scope=namespace	Threshold: scope=cluster
#Nodes	n/a	5000
#Namespaces	n/a	10000
#Pods	3000	150000
#Pods per node	min(110, 10*#cores)	min(110, 10*#cores)
#Services	5000	10000
#All service endpoints	TBD	TBD
#Endpoints per service	250	n/a
#Secrets	TBD	TBD
#ConfigMaps	TBD	TBD
#Deployments	2000	TBD
#DaemonSets	TBD	TBD
#Jobs	TBD	TBD
#StatefulSets	TBD	TBD
#AccessTokens	2000	2000
#AccessTokens verifications	5000 QPS	5000 QPS

aws

Reliability - Scalability

Cluster Auto Scaler

Cluster Autoscaler makes sure that all pods in the cluster have a place to run, no matter if there is any CPU load or not. Moreover, it tries to ensure that there are no unneeded nodes in the cluster.
CPU-usage-based (or any metric-based) cluster/node group autoscalers don't care about pods when scaling up and down. As a result, they may add a node that will not have any pods, or remove a node that has some system-critical pods on it, like kube-dns. Usage of these autoscalers with Kubernetes is discouraged.

Prewarm

if your cluster workload doesn’t have a consistent usage and scale in/out in a high spike fashion then it is possible to overload the EKS control plane so you will face performance degradation. If you are planning to run workloads that requires high spike loads such as ML data training in which you could be scaling your cluster with spikes of ~1000 pods then you should engage your TAM and Premium Support in order to pre-scale your EKS control plane.

aws

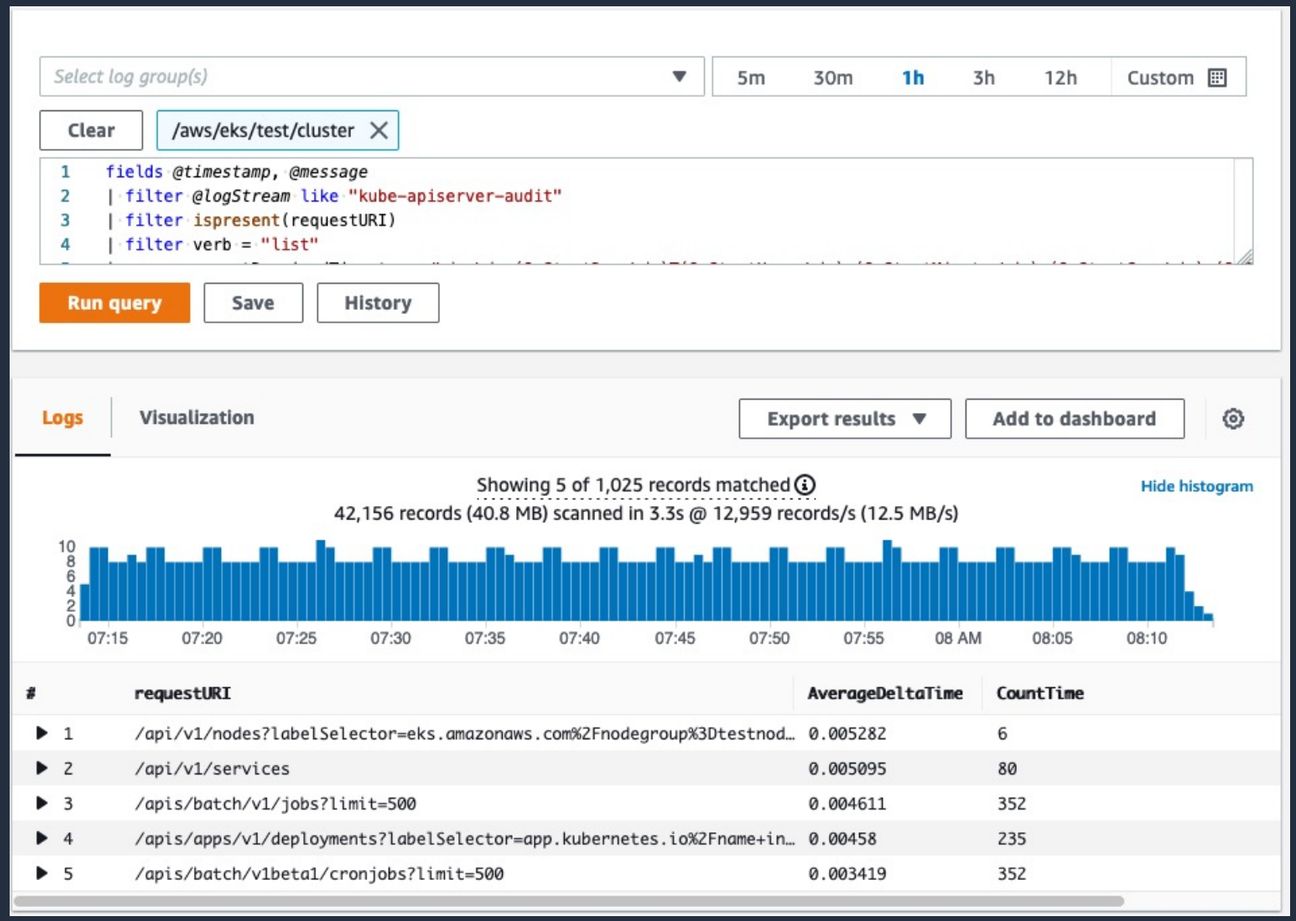
Reliability - Scalability

API latency

1. List operations overall. These tend to be more “expensive” API operations

2. Check the average response latency.

3. Look for an unusually high number of requests. Anything with more requests than the kubelet is suspect. Check for an unusual number of queries from common components like CoreDNS, metrics servers, or HPA/VPA.



aws

Reliability - Networking

DNS Scaling

By default, an EKS cluster runs two CoreDNS replicas; this is the minimum you should run in order to maintain high availability for DNS requests. As a reasonable starting point, **a single CoreDNS replica can support up to 16 worker nodes, or 256 CPU cores** (whichever is smaller). This is not a hard rule, but a good place to start when you look at scaling up your cluster; you may need to scale differently for your cluster's workload.

Kubernetes has several ways to provide autoscaling for pods, and either Cluster Proportional Autoscaler (CPA) or Horizontal Pod Autoscaler (HPA) will work well with CoreDNS. CPA lets you scale CoreDNS directly against the number of cores and worker nodes in your cluster while HPA can scale based on the CPU and memory utilization of the CoreDNS pods themselves.

DNS Packet Limit

The VPC resolver can accept only a maximum hard limit of 1024 packets per second per network interface. If more than one CoreDNS pod is on the same worker node, or in case of excessive dns request on a single node, then the chances of hitting this limit are higher for external domains queries. Here is the more details on how to review your current workload against this limitation:

aws

Reliability - Networking

DNS ndots Config

The ndots value is the number of dots that must appear in a name to resolve a query before an initial absolute query is made. To prevent unnecessary dns queries, our recommendation to configure and test ndots value of 2 rather than 5 which is the default value.

// /etc/resolv.conf

nameserver 10.100.0.10

search default.svc.cluster.local svc.cluster.local cluster.local ap-northeast-2.compute.internal

options ndots:5

default.svc.cluster.local: a helper domain to discover Kubernetes **services in the same namespace as your pod**.

svc.cluster.local: a helper domain to discover Kubernetes **services in other namespaces.**

cluster.local: cluster.local is the **default domain name for your EKS cluster.**

ap-northeast-2.compute.internal: This search domain is added to the pod by the kubelet, as it is part of the /etc/resolv.conf file on the EKS AMI. This is because the default setting for a pod's dnsPolicy configuration is to inherit name resolution from the worker node that the pod is running on.

database-1.ap-northeast-2.rds.amazonaws.com

database-1.ap-northeast-2.rds.amazonaws.com.default.svc.cluster.local

database-1.ap-northeast-2.rds.amazonaws.com.svc.cluster.local

database-1.ap-northeast-2.rds.amazonaws.com.cluster.local

database-1.ap-northeast-2.rds.amazonaws.com.ap-northeast-2.internal

aws

Reliability - Networking

CoreDNS Tuning

In this configuration, **we've setup a server block for cluster.local to handle DNS lookups that are internal to the cluster, and the . server block will take care of any other requests.** The search domains that enable service discovery in the EKS cluster all end with cluster.local, so they should be handled by the CoreDNS kubernetes plugin. Since that plugin already uses in-memory storage for addresses, we've removed the cache for these internal lookups. Finally, we've enabled the lameduck setting for the health plugin, as this will help reduce DNS lookup failures if CoreDNS is redeployed. This setting tells CoreDNS how long to wait after startup before registering itself as healthy, setting it to 5 seconds lets CoreDNS fully come online before it starts accepting requests.

Default CoreDNS Corefile

```
.:53 {
    errors
    health
    kubernetes cluster.local in-addr.arpa ip6.arpa {
      pods insecure
      fallthrough in-addr.arpa ip6.arpa
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
}
```

Proposed CoreDNS Corefile

```
cluster.local:53 {
      errors
      log
      health {
        lameduck 5s
      }
      ready
      kubernetes {
        fallthrough in-addr.arpa ip6.arpa
      }
      prometheus :9153
      reload
      loop
      forward . /etc/resolv.conf
      loadbalance
    }
    . {
      errors
      prometheus :9153
      forward . /etc/resolv.conf
      cache
      loop
    }
```


aws

Reliability - Networking

VPC & Subnet

VPC: 10.0.0.0/24

3 EKS Subnets /27: Used for EKS Worker Nodes, we use here 1 Managed Node Group across the 3 subnets.

/27 Subnet = 30 Usable IPs
-3 IP reserverd per subnet = 27
*3 Subnets = 81 IPs
-2 EKS Control Plane = 79 IP addresses for pods.

Each pod that you deploy is assigned one secondary private IP address from one of the network interfaces attached to the instance. Previously, it was mentioned that an m5.large instance supports three network interfaces and ten private IP addresses for each network interface. Even though an m5.large instance supports 30 private IP addresses, you can't deploy 30 pods to that node. To determine how many pods you can deploy to a node, use the following formula:

(Number of network interfaces for the instance type × (the number of IP addressess per network interface - 1)) + 2

aws

Reliability - Networking

CNI WARM_ENI_TARGET

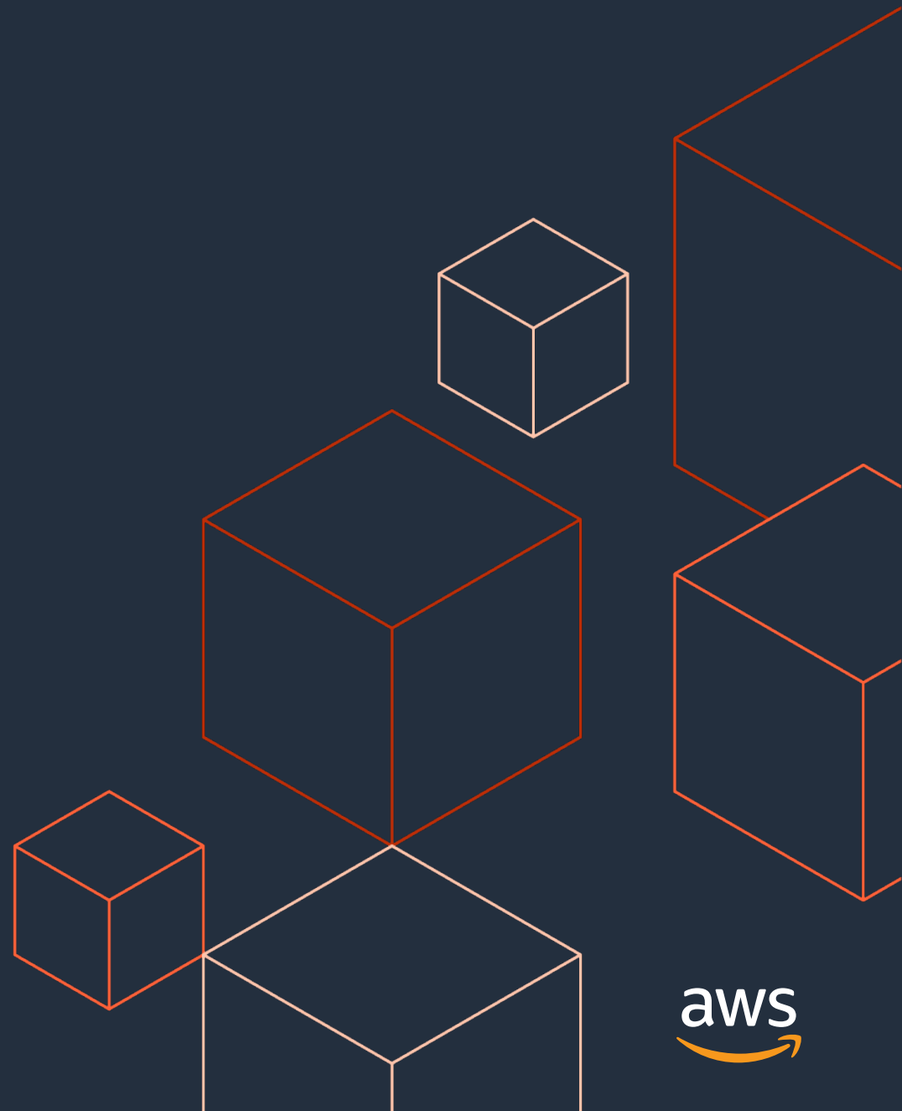
Instance type	WARM_ENI_TARGET	WARM_IP_TARGET	MINIMUM_IP_TARGET	Pods	Attached ENIs	Attached Secondary IPs	Unused IPs	IPs per ENI
t3.small	1	-	-	0	1	3	3	3
t3.small	1	-	-	5	3	9	4	3,3,3
t3.small	1	-	-	9	3	9	0	3,3,3
t3.small	-	1	1	0	1	1	1	1
t3.small	-	1	1	5	2	6	1	3,3
t3.small	-	1	1	9	3	9	0	3,3,3
t3.small	-	2	5	0	2	5	5	2,3
t3.small	-	2	5	5	2	7	2	3,3,1
t3.small	-	2	5	9	3	9	0	3,3,3
p3dn.24xlarge	1	-	-	0	1	49	49	49
p3dn.24xlarge	1	-	-	3	2	98	95	49,49
p3dn.24xlarge	1	-	-	95	3	147	52	49,49,49
p3dn.24xlarge	-	5	10	0	1	10	0	10
p3dn.24xlarge	-	5	10	7	1	12	5	12
p3dn.24xlarge	-	5	10	15	1	20	5	20
p3dn.24xlarge	-	5	10	45	2	50	5	49,1

https://github.com/aws/amazon-vpc-cni-k8s/blob/a8bea42d0344f66e7cbfb67407db3cdbd8335f71/docs/eni-and-ip-target.md

aws

Application migration

Rearchitecting for MSA adoption



aws

Things to consider when migrating application

- MSA Core Concept
    - Decomposition
    - Data Management
- Rearchitecting
    - SLA
        - Non Functional Requirements
        - Performance, Availability
        - Cost, Maintainability
    - Extendibility
        - Scale Cube
    - Service migration
        - MSA Core Concept: Refactoring

aws

Process of Application Rearchitecting

- Decomposition
  - Review and Analysis
- Data Management

- Decomposition
  - Core
  - Supporting
  - Generic
- Data Management
  - Database per Service
  - Shared database

- Refactoring

aws

Requirement Workshop is...

- Define Purpose and Goal
    - Non Functional Requirement for Rearchitecting
        - Cost Down
        - Extendibility, Maintainability Enhance
- Review old architecture
    - Decomposition Plan
    - Data Management Plan

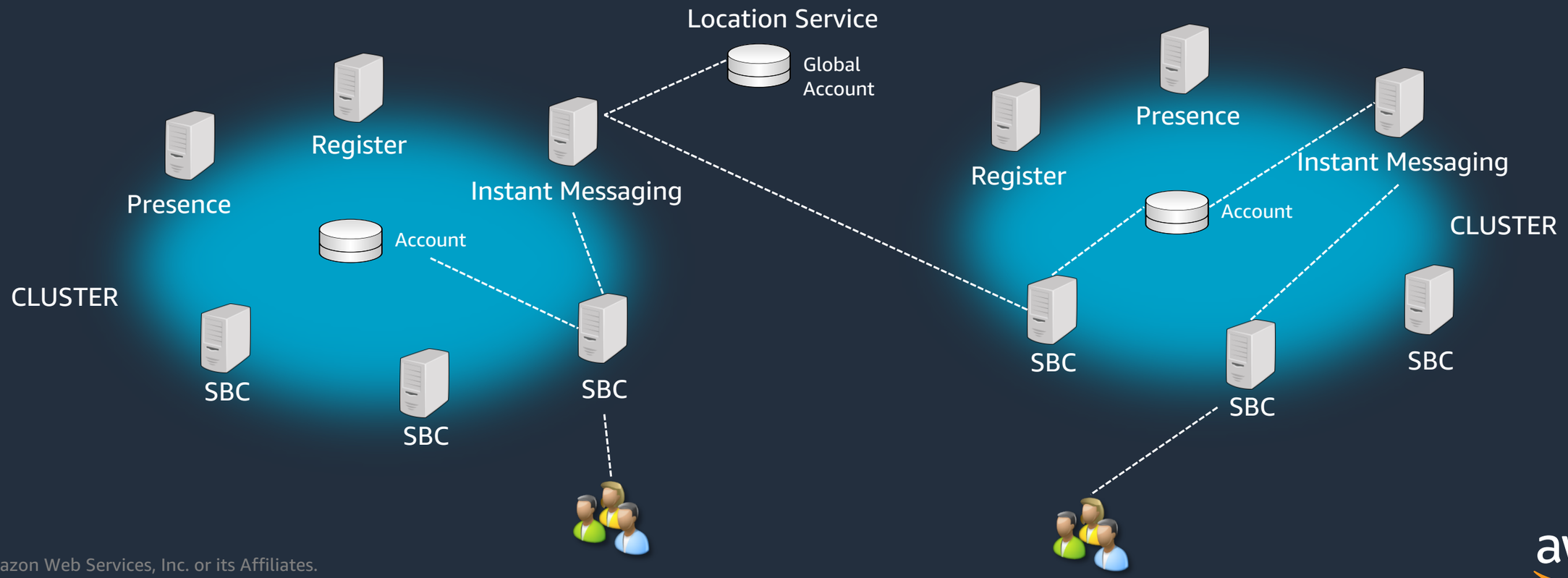
aws

Requirement Workshop - Review old architecture

SAMSUNG RCS1.0

- Session: Connection for messaging service between users
- SBC(Session Boarder Controller): Connection Broker
- Instant Messaging: Application Server for sending/receiving messages
- Account: User Profile for each cluster
- Global Account: Index for Account

- 28,000,000 users
- 1,000,000 users/cluster
- 10~20 cluster for each operators
- x2 resources for backup cluster

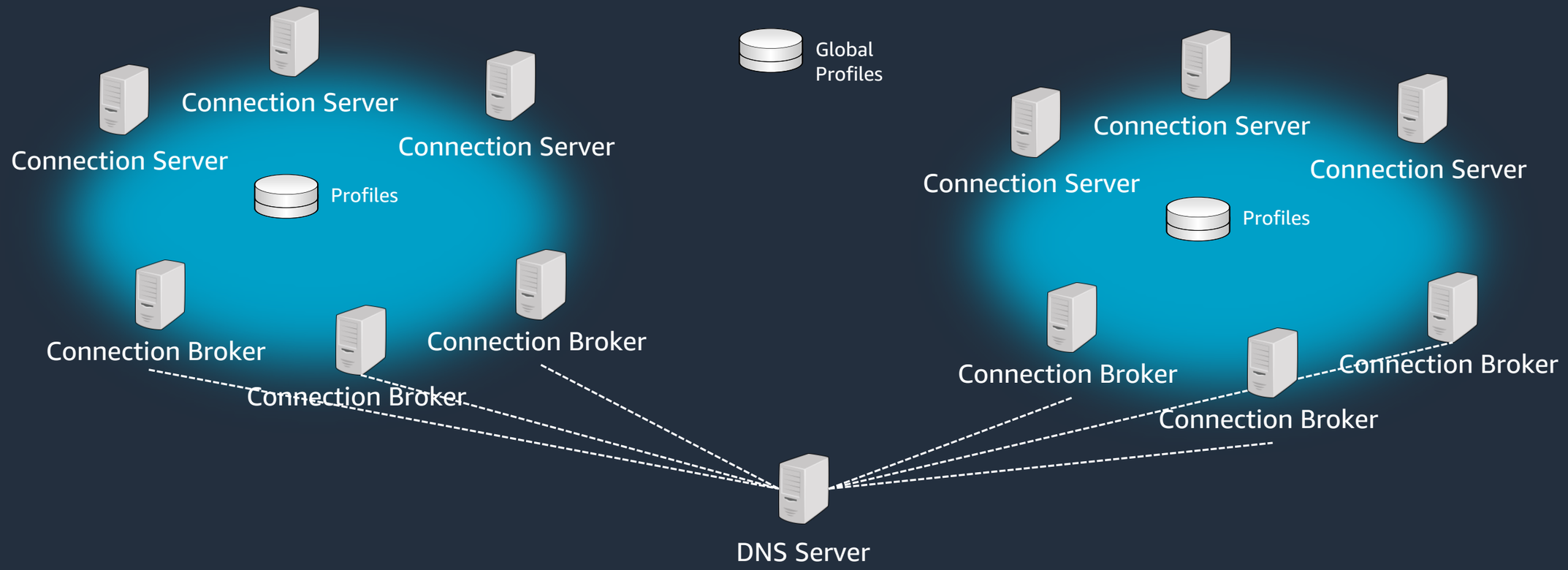


aws

Requirement Workshop - Review old architecture

Monolith Connection Broker Architecture

- Cluster and Global Profiles



Analysis and Evaluation Architecture is...

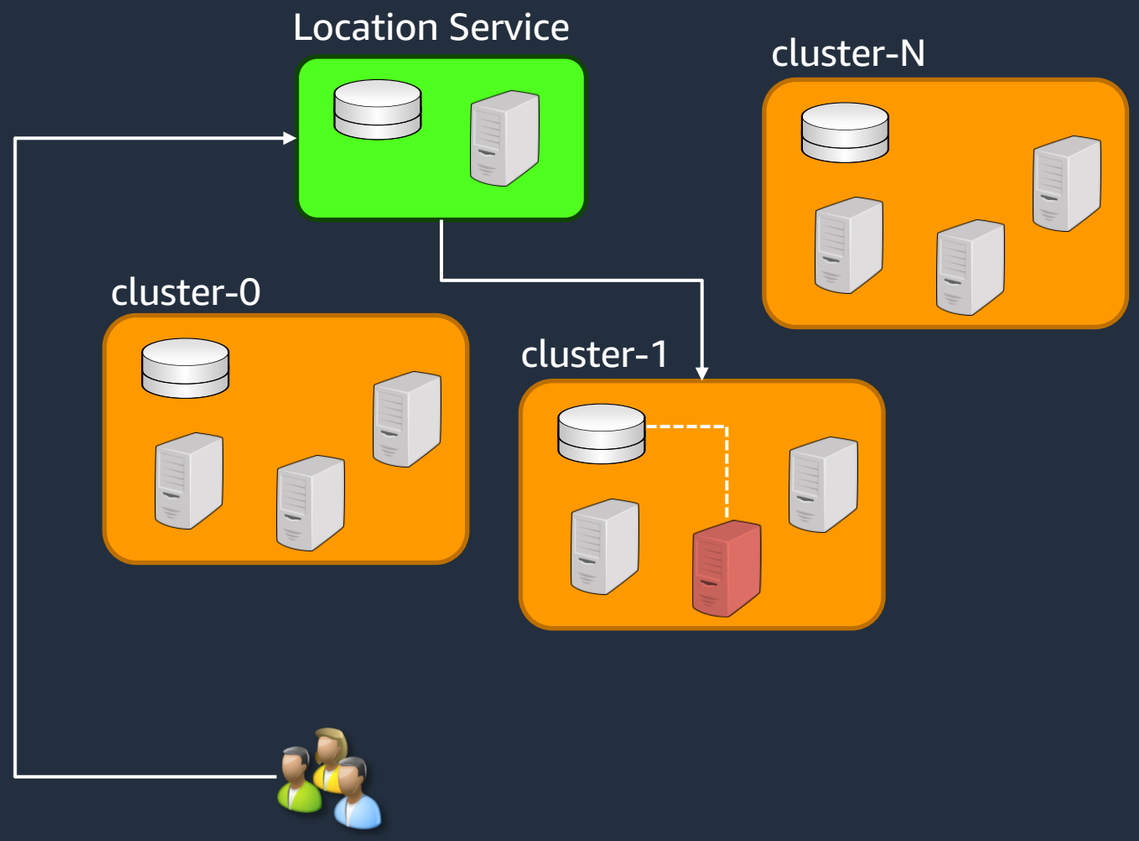
- Decomposition Plan
    - Figure out obstacles for scale cube
        - For Extendibility, Maintainability
    - Figure out waste of resources
        - For Cost down
- Data Management Plan
    - Figure out Data Management Limitation
        - For Extendibility

aws

Analysis and Evaluation Architecture

Figure out limitation for scale in/out

- Applications depend on data



Limits Cluster-size
- Capacity of data repository
- Physical limitation
- Security issue

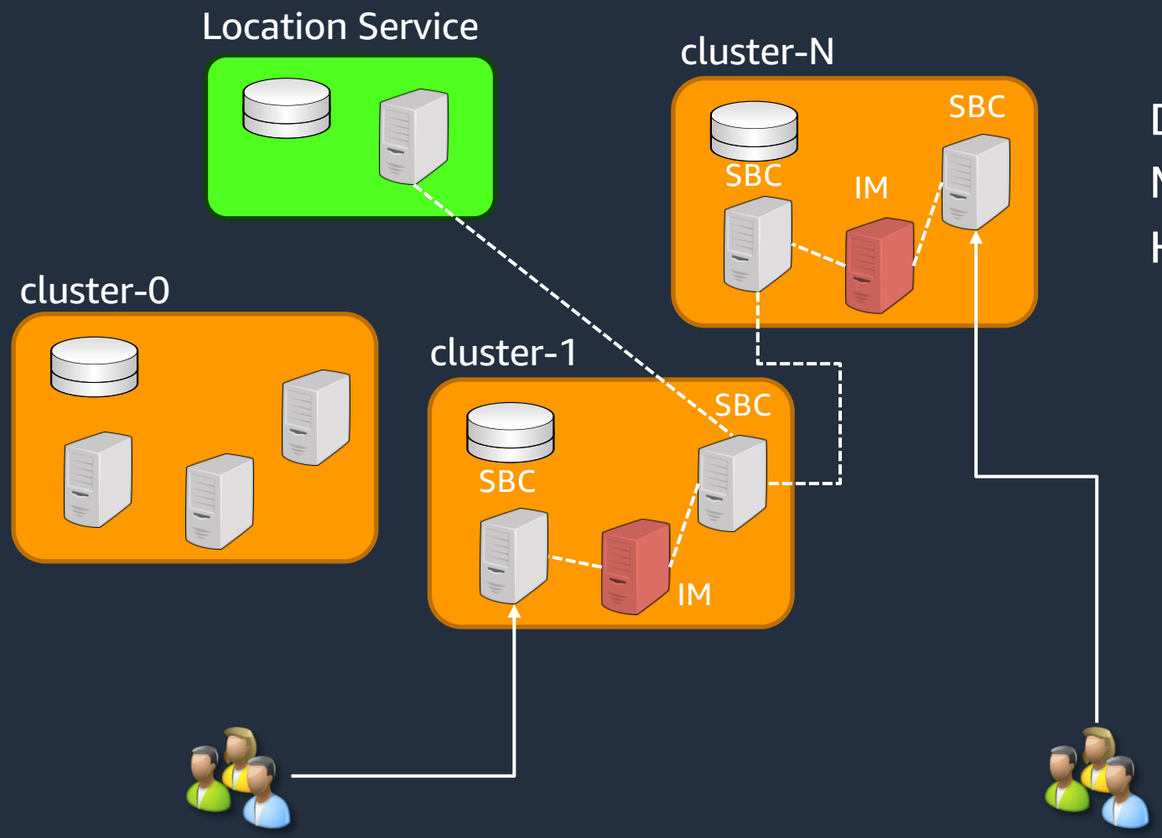
Disadvantage for scale in/out
- Data is stored in cluster repository permanently
- Cluster resources are wasted most of the time

aws

Analysis and Evaluation Architecture

Figure out limitation for scale out

- Stateful Application



Dependency make steps of discovery and relay
Module have to carry processing data
Hard to recover processing data at accident

aws

Rearchietcting is…

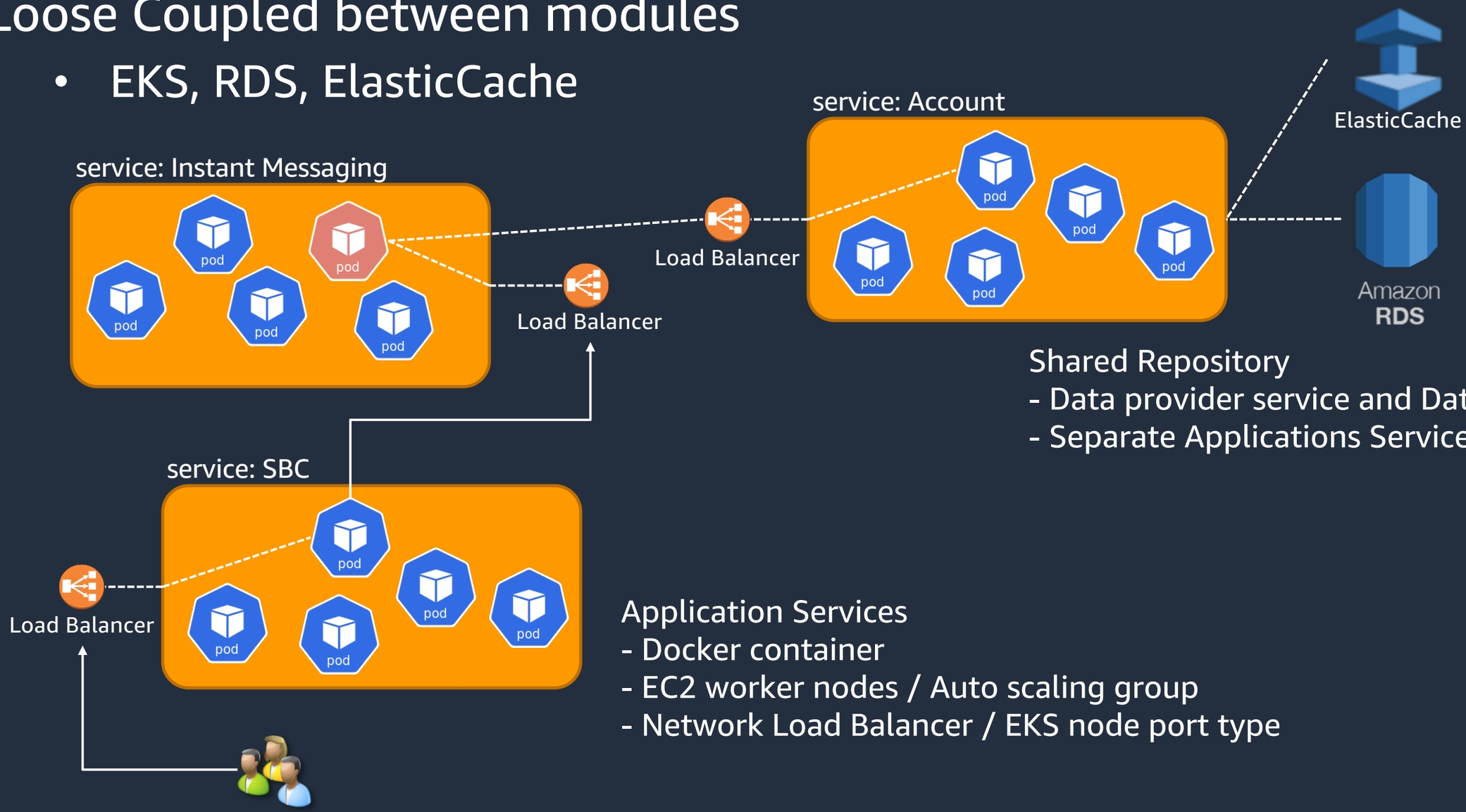
- Decomposition
    - Decompose by Subdomain
        - Core(Instant Messaging), Supporting(Presence), Generic(Register)
        - EKS service mapped to subdomains
    - Scale cube
        - Scale: Traffic distribution
        - Functionality: Service Module
- Data Management Plan
    - Database per Service
        - Data as service
        - Data provider service
    - API Composition

aws

Application Rearchitecting

Loose Coupled between modules

- EKS, RDS, ElasticCache

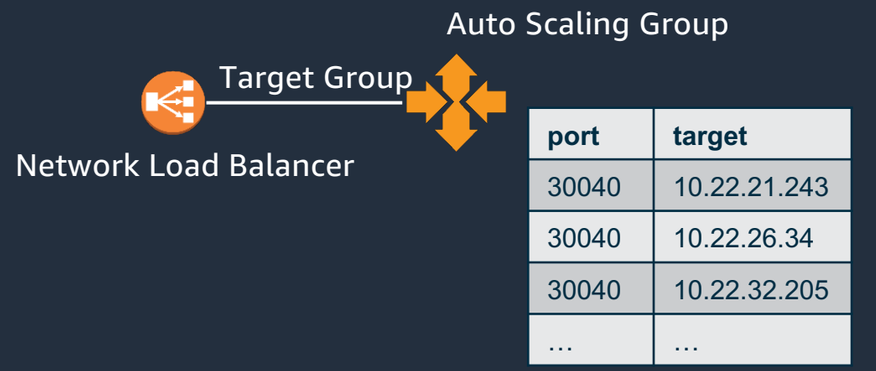


aws

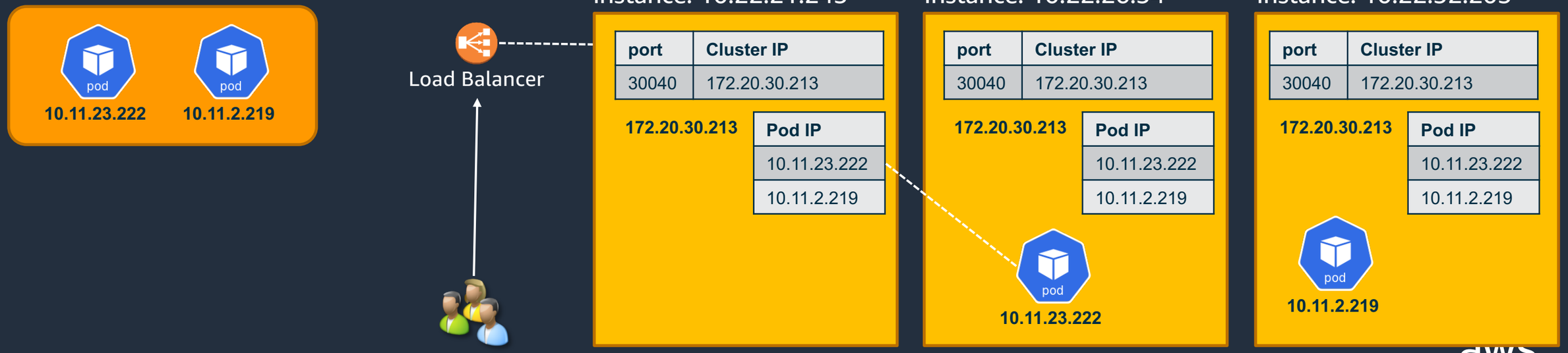
Application Rearchitecting

Stateless Application Service

- NLB
- Node port service type



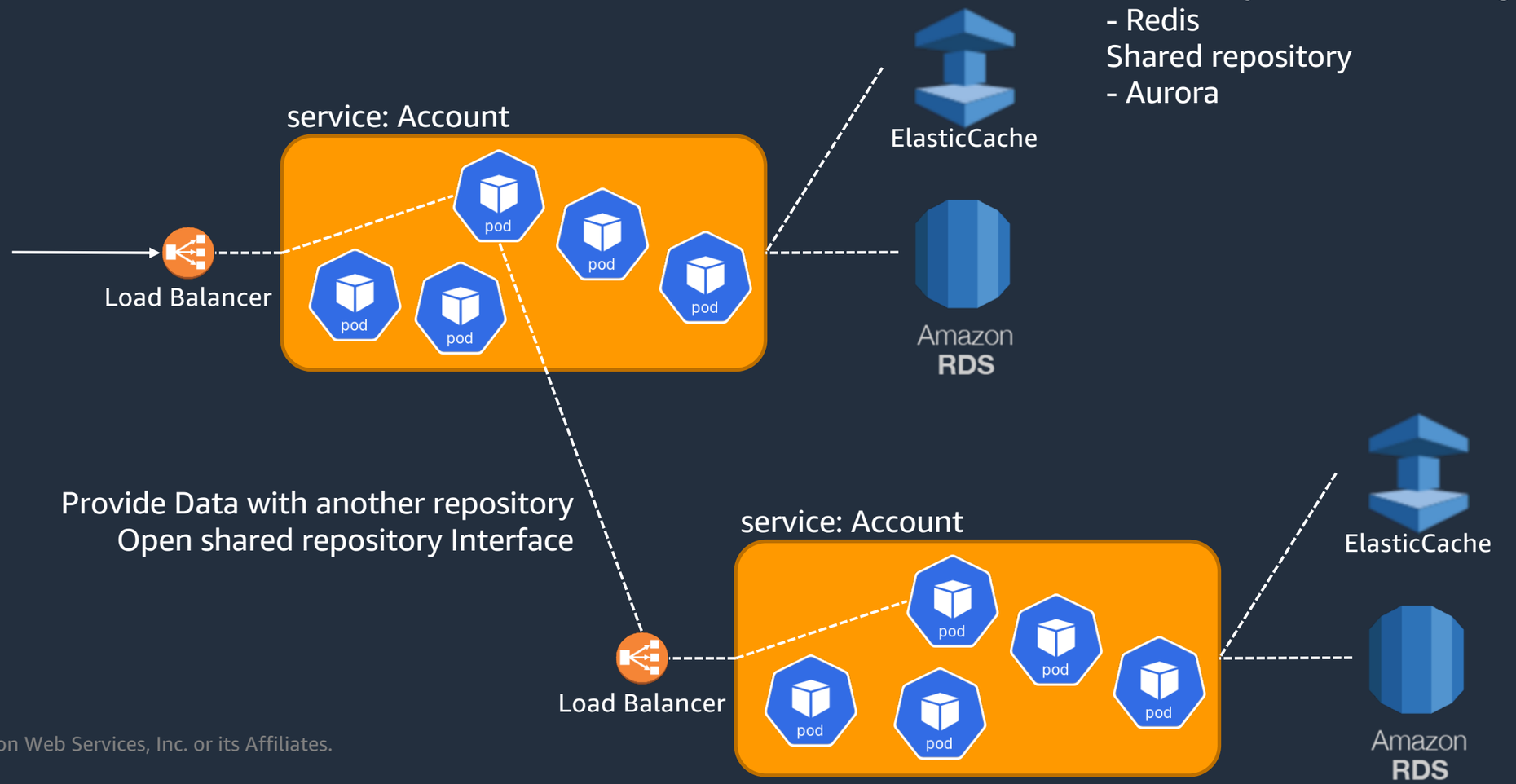
service: SBC
NodePort type
- NodePort 30040
- ClusterIP 172.20.30.213



Application Rearchitecting

Service independent with data

- Data Provider Service
- API Composition

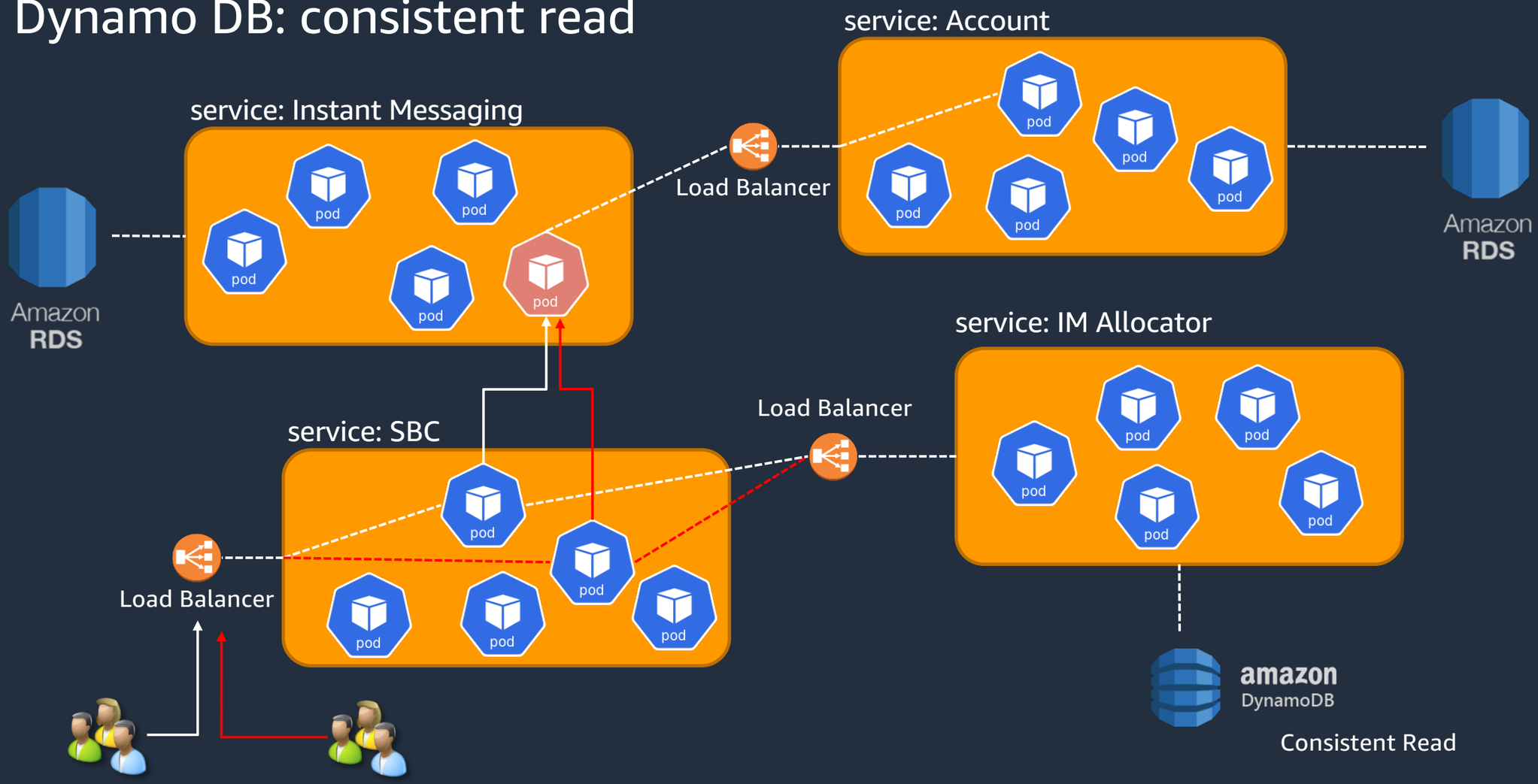


aws

Application Rearchitecting

Discovery with consistency

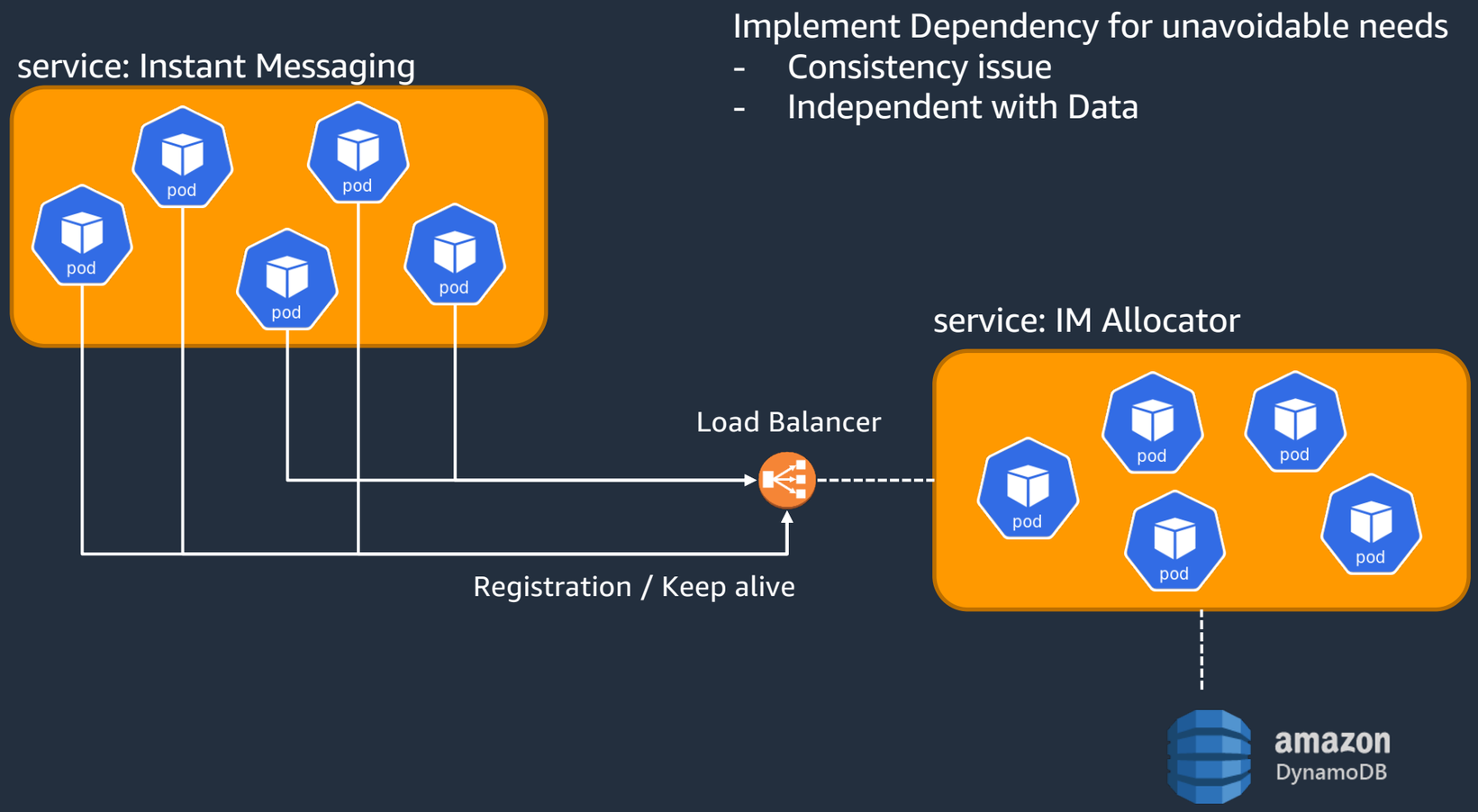
- Dynamo DB: consistent read



Application Rearchitecting

Discovery with consistency

- Registration and health check

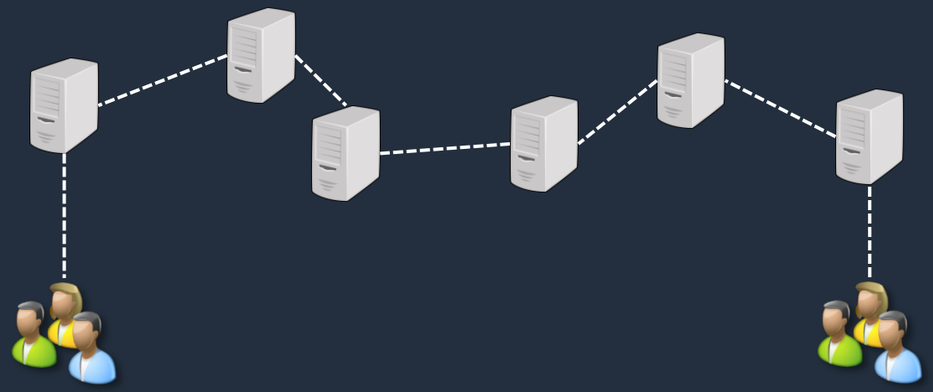


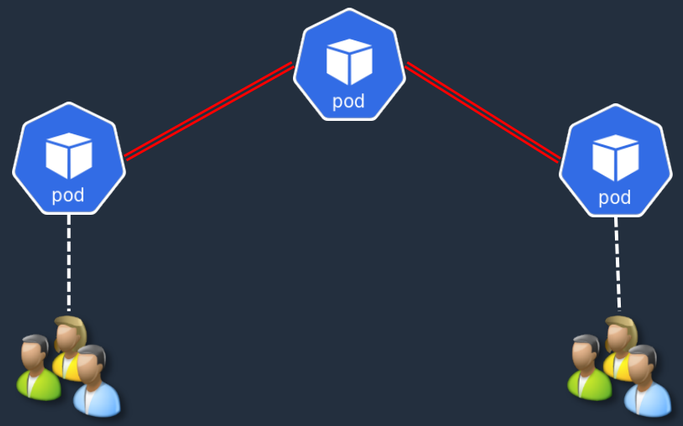
aws

Application Rearchitecting

High performance communication channel

- gRPC





	RESTfull	gRPC
Protocol	HTTP	HTTP2
Payload	JSON	Protobuf
Prescriptiveness	Loose. Any HTTP is valid	Strict specification
Streaming	Client, Server	Client, Server, Bi-directional
Security	TLS	TLS

aws

LB for gRPC

NLB with client load balancer

ALB Controller

aws

Migration & Rebalancing

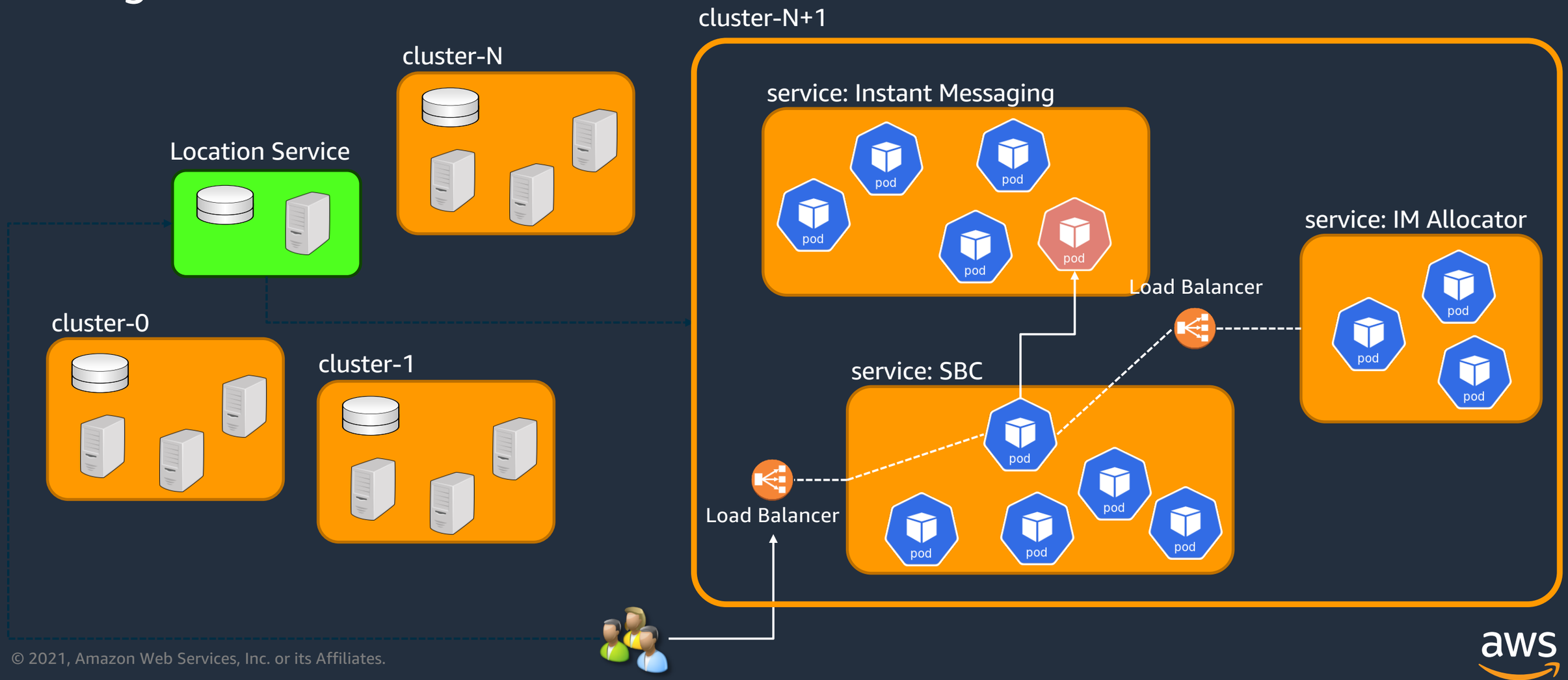
- Refactoring
    - Stranger Application
- Migration
    - Going live
        - Seamless
        - Continuous service for users
    - Cutover
        - Easy to Fallback

aws

Service Rebalancing

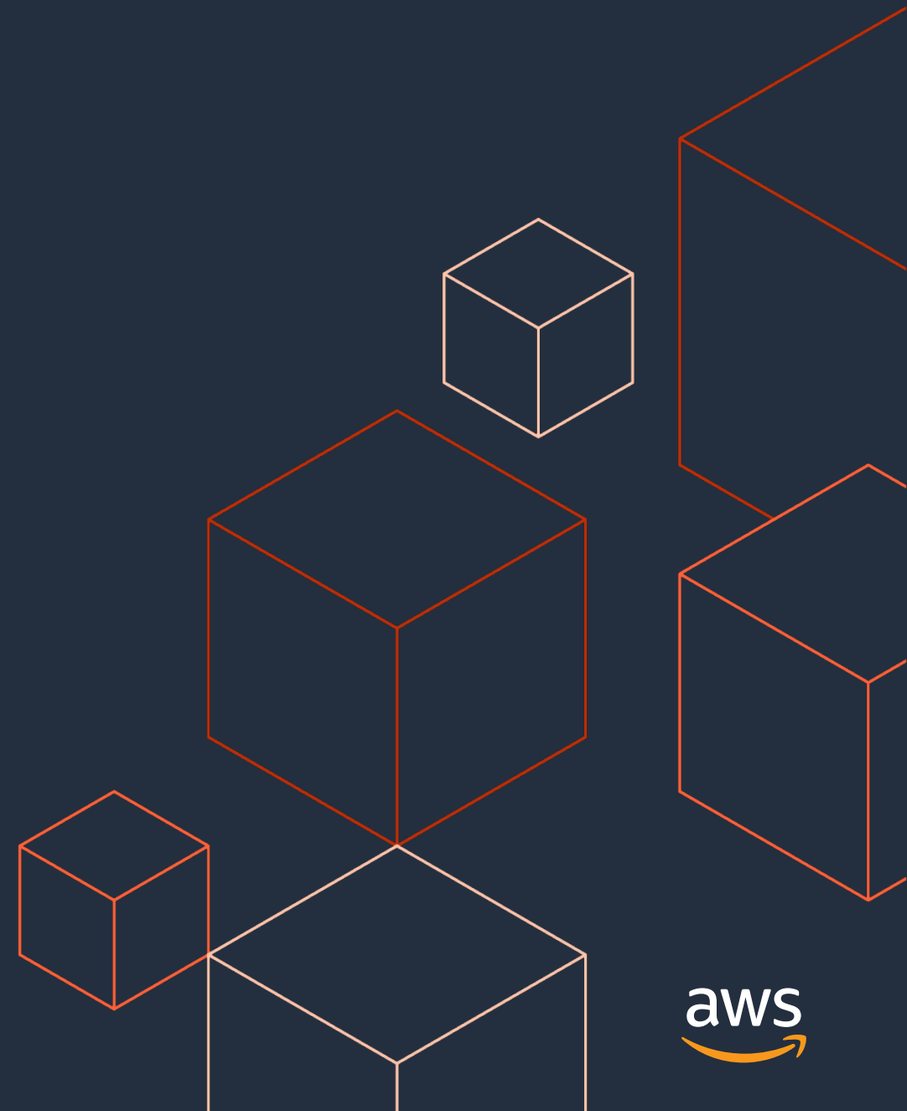
Refactoring: Stranger Application

Going live



aws

Conclusion



Key thing to adopt MSA

- Organization - Rather than arbitrarily amalgamating teams that seek high levels of operational efficiencies in their infrastructure and organizations that seek to implement business logic, they share their powers and approaches in line with each organization's goals.

- Establishment of working process for each team - Even if the operation team and the development team are not tightly integrated with each other in order to carry out the goals of each team, they can use each other's output through the process to provide services.

Reference

EKS Best Practices – https://aws.github.io/aws-eks-best-practices/

K8s Troubleshooting - https://learnk8s.io/a/a-visual-guide-on-troubleshooting-kubernetes-deployments/troubleshooting-kubernetes.v2.pdf

K8s Testing - https://static.sched.com/hosted_files/kccna18/9b/Kubecon%20Seattle%20SIG-Testing%20Deep%20Dive%20%281%29.pdf

K8s Scalability - https://docs.google.com/document/d/1hEpf25qifVWztaeZPFmjNiJvPo-5JX1z0LSvvVY5G2g/edit

EKS CDK Quick Start (in Python) - https://github.com/aws-quickstart/quickstart-eks-cdk-pytho

aws

Refrence for Security – Pod Security

- kube-psp-advisor is a tool that makes it easier to create K8s Pod Security Policies (PSPs) from either a live K8s environment or from a single .yaml file containing a pod specification (Deployment, DaemonSet, Pod, etc).
- open-policy-agent/gatekeeper-library: The OPA Gatekeeper policy library a library of OPA/Gatekeeper policies that you can use as a substitue for PSPs.
- A collection of common OPA and Kyverno policies for EKS.
- Policy based countermeasures: part 1
- Policy based countermeasures: part 2

aws

Refrence for Security – Multi Tenancy Tools and Resource

Banzai Cloud

Kommander

Lens

Nirmata

Rafay

Rancher

Weave Flux

aws

Refrence for Security – Detective Control

- kubeaudit
- MKIT
- kube-scan Assigns a risk score to the workloads running in your cluster in accordance with the Kubernetes Common Configuration Scoring System framework
- amicontained Reveals which Capabilities are allowed and syscalls that are blocked by the container runtime
- kubesec.io
- polaris
- Starboard
- kAudit
- Snyk

aws

Refrence for Security – Runtime Resources

- 7 things you should know before you start
- AppArmor Loader
- Setting up nodes with profiles
- zaz A command-line tool to assist on assessing container security requirements and generating seccomp profiles
- seccomp-operator Is similar to the AppArmor Loader, only instead of AppArmor profiles, it creates a seccomp profiles on each host

aws

Refrence for Security – Runtime Resources Tools

- Aqua
- Qualys
- Stackrox
- Sysdig Secure
- Twistlock

aws

End of Document