# Security for Cloud Native Workloads

22-May-2021 | DevSecOps Conf 2020

Runcy Oommen

# |In brief|

- ✓Generic cloud native & cloud security overview
- ✓Shared responsibility model
- ✓Microservices security
- ✓Containers v/s Serverless
- ✓Securing the container lifecycle
- ✓Kubernetes security improvements

# [~]$ whoami

## Career

- Principal SDE, SONICWALL, 18+ yrs. industry experience primarily in systems, cloud (private/public), security, networking
- 10x multi-cloud certified (GCP, AWS, Azure, CNCF)
- Patent (India) in cloud security around distributed data storage
- Interested in serverless, containers and cloud native offerings. Firm believer of a multi-hybrid cloud future

## Community

- Organizer of GDG Cloud, AWS user Group and Cloud Native meetup groups in Bangalore
- Regular speaker at domestic and international cloud, tech & security conferences
- Multiple hackathon wins in cloud/security topics.
- Recognized by Google as a community influencer

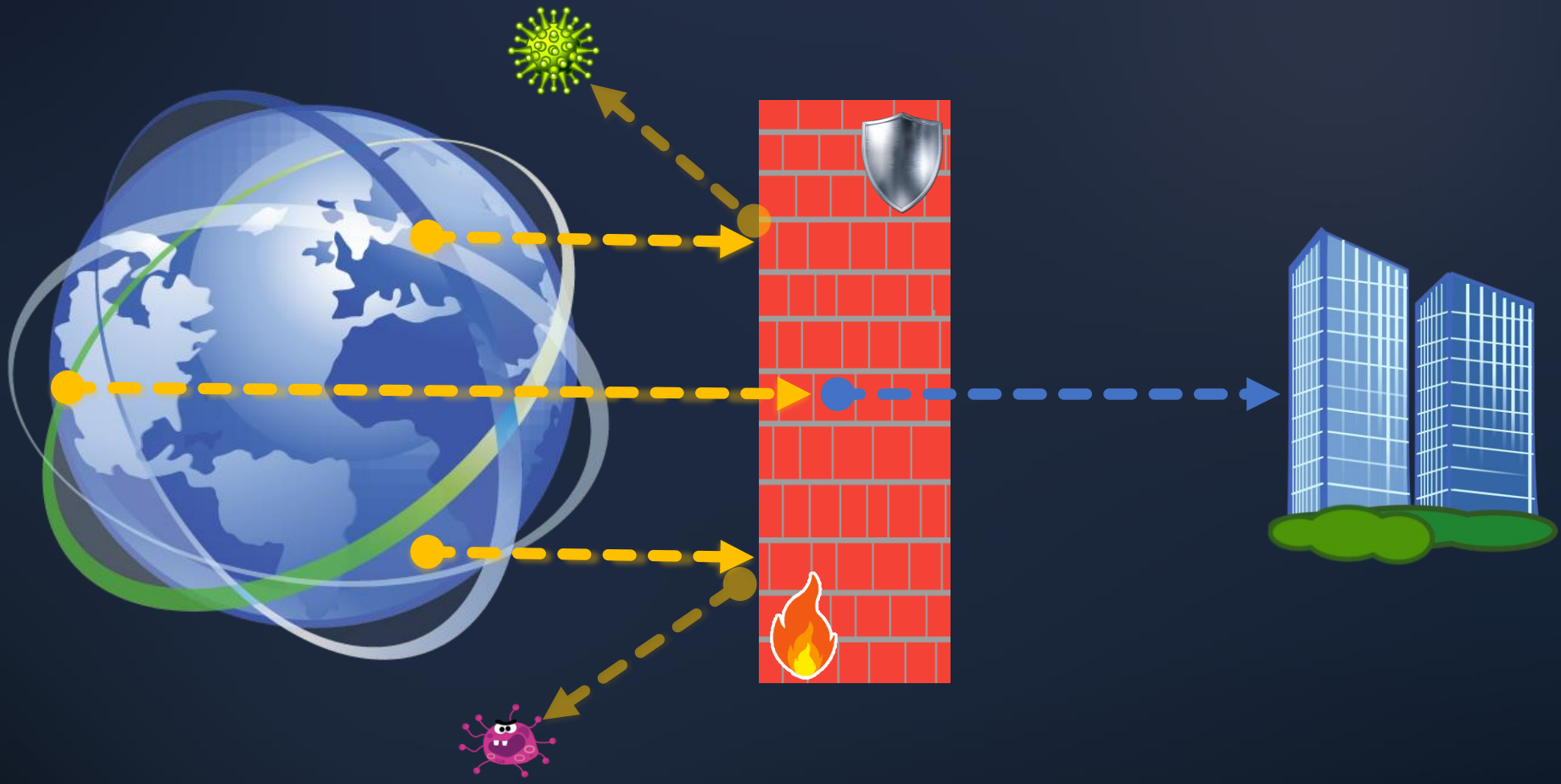
runcyoommen
https://runcy.me

# What does “Cloud Native” mean?

**A cloud computing approach to build & run scalable apps in modern environments such as public, private and hybrid clouds. Technologies such as containers, microservices, serverless and immutable infrastructure, deployed are elements of this architecture.**

**Reference:**
https://en.wikipedia.org/wiki/Cloud_native_computing


IT infrastructure & landscape has undergone a paradigm shift…



# Traditional view

runcyoommen


Modern view


So, how exactly should cloud native security differ from traditional network security?

# Important facets of cloud

## Ubiquitous

The cloud is always reachable from anywhere, any time, any device

## Integrated

Security and other services talk to each other for full visibility

## Intelligent

The cloud learns from every user, network and traffic

## Scalable

You can add new features and support users without breaking a sweat

## Comprehensive

The Cloud scans every byte – ingress and egress – including SSL & CDN

## Available

Capabilities can be guaranteed to remain accessible

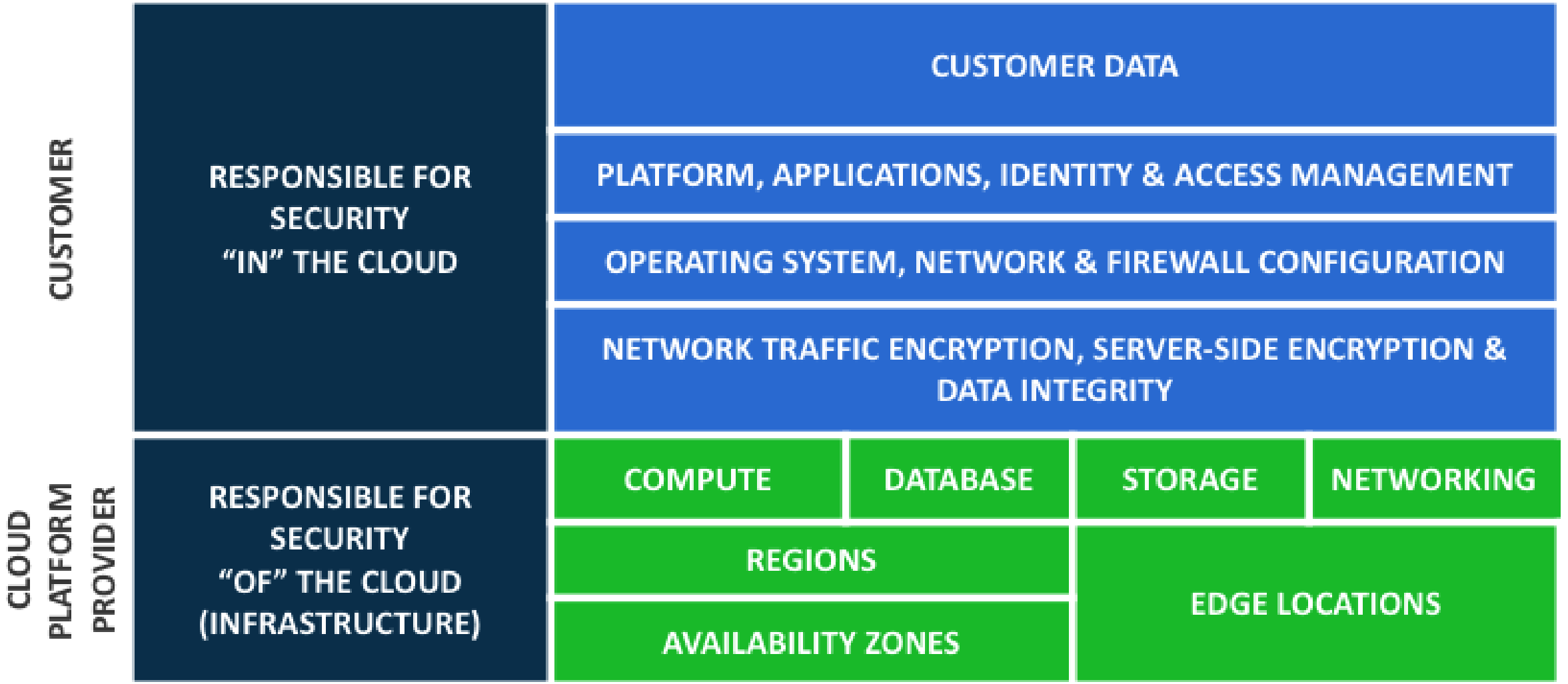
runcyoommen

# Cloud Features v/s Security Balances

- Agility
- Self service
- Scale
- Automation
- Pay as you go

- Gatekeeper
- Standards
- Control
- Centralized
- Timely alerts

# Shared Responsibility Model

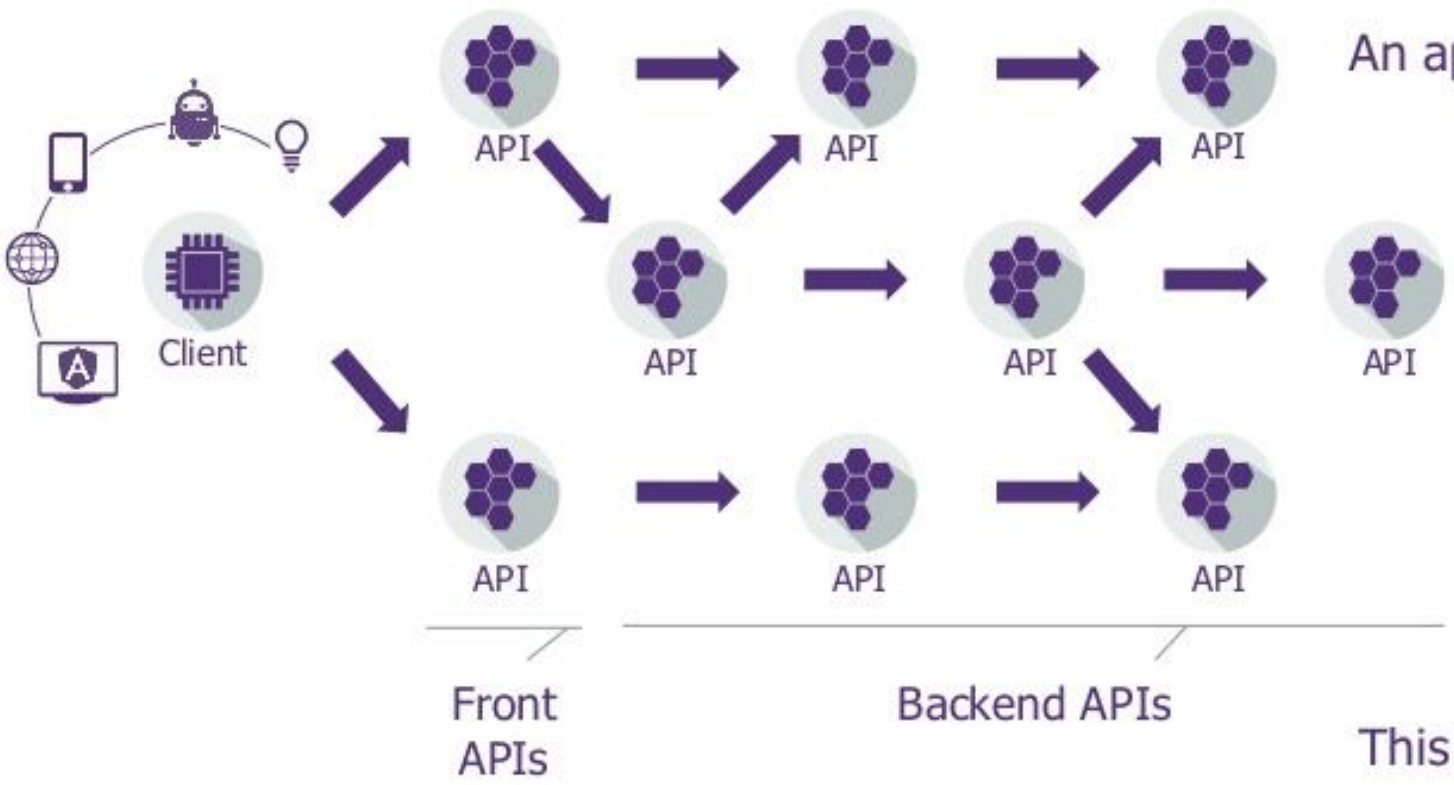


# Let’s begin the journey…

from
# MONOLITH

to
M-I-C-R-O-S-E-R-V-I-C-E-S

# What I mean when I say « microservices »




An application calling an API endpoint…

…calling another API endpoint

…calling another API endpoint

…calling other API endpoints

…

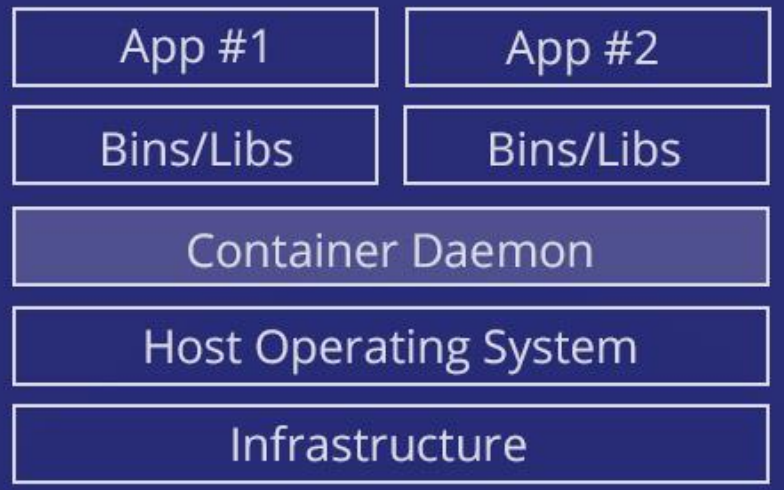
This generally also involves CI/CD tools

and various degrees of automation

## VIRTUAL MACHINES


WHATS *the* DIFF?

## CONTAINERS




VM
Container

# How Containerization Helps?

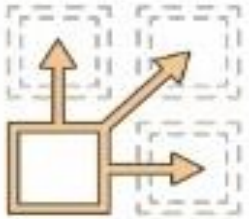
- **Portability** – By abstracting applications from host, it's easier to run on any platform or cloud
- **Scalability** – Containerized applications holds the ability to handle increasing workloads
- **Fast Development** – Allows developers to change and track changes in the platform's source code ensuring high productivity
- **Security** – App isolation ensures that separate containers run independently
- **Continuity** – Failure of one will not influence the state of others
- **Easy Management** – Orchestration performs application management, automated installation and workload management

runcyoommen

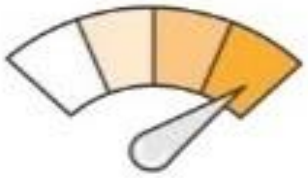
# What is serverless?

Build and run applications
without thinking about servers

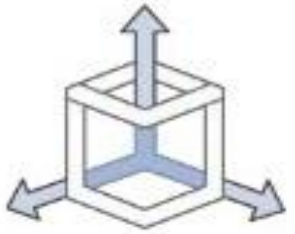
## Removes the need for...



Provisioning and Utilization



Operations and Management



Scaling

Availability and Fault Tolerance

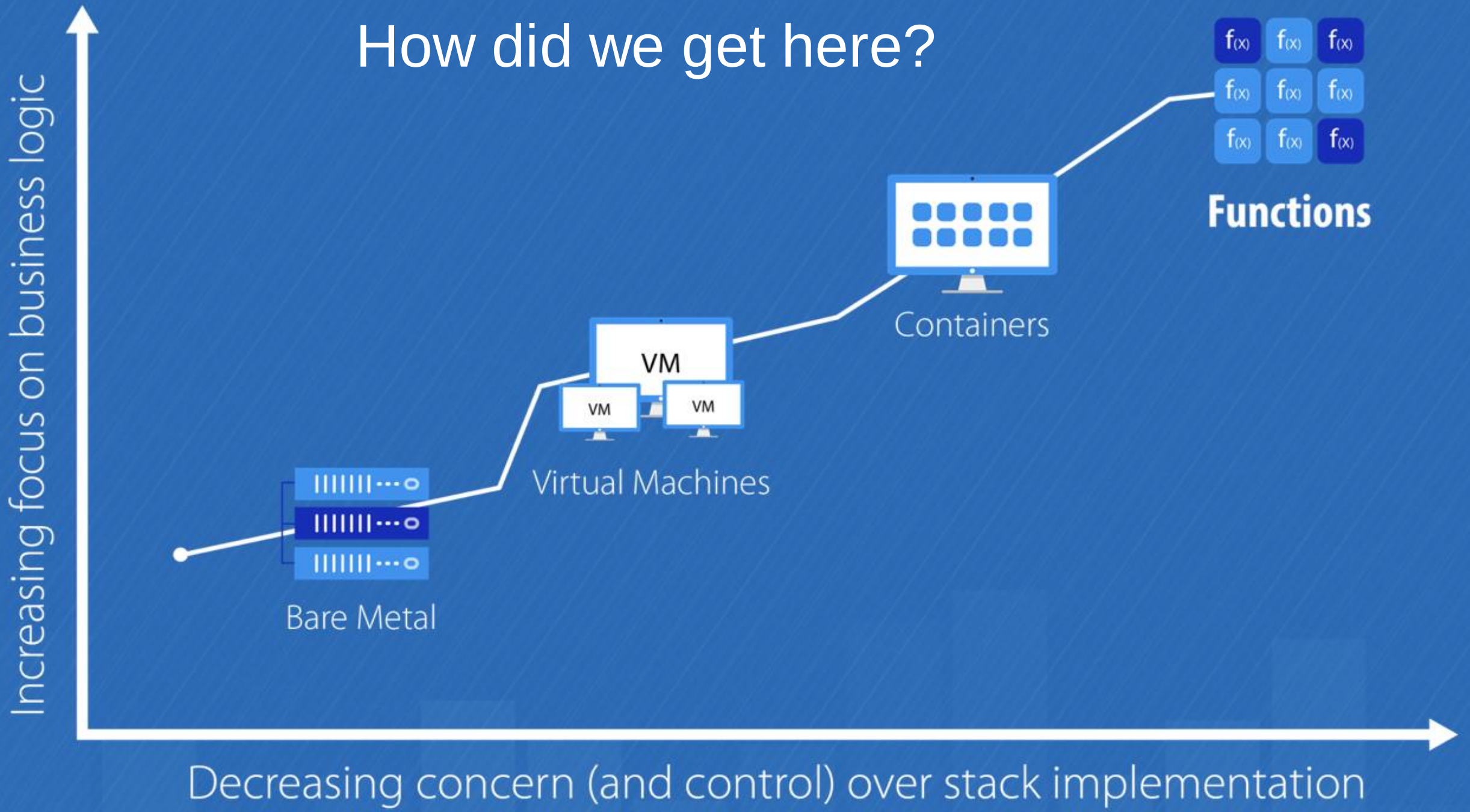
## Provides these...

Abstraction of servers


Event-driven/ instant scale

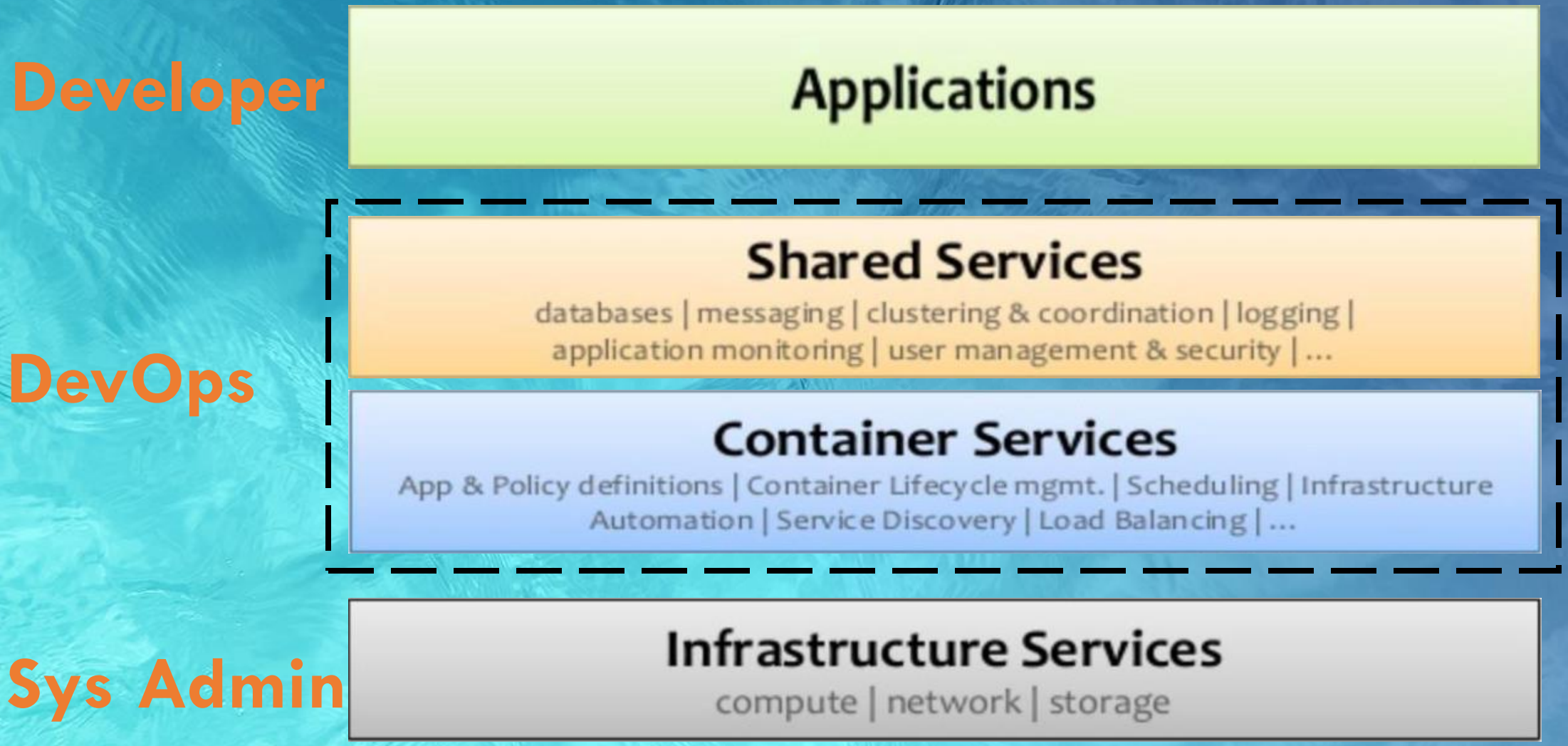
Sub-second billing

# How did we get here?

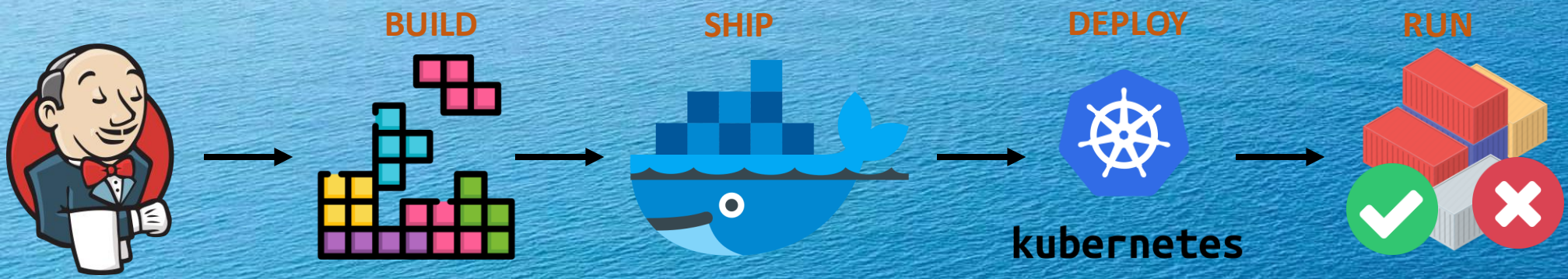



Serverless is
Eating the Stack

# The stack of *SECURITY* responsibilities…



# SECURING THE CONTAINER LIFECYCLE

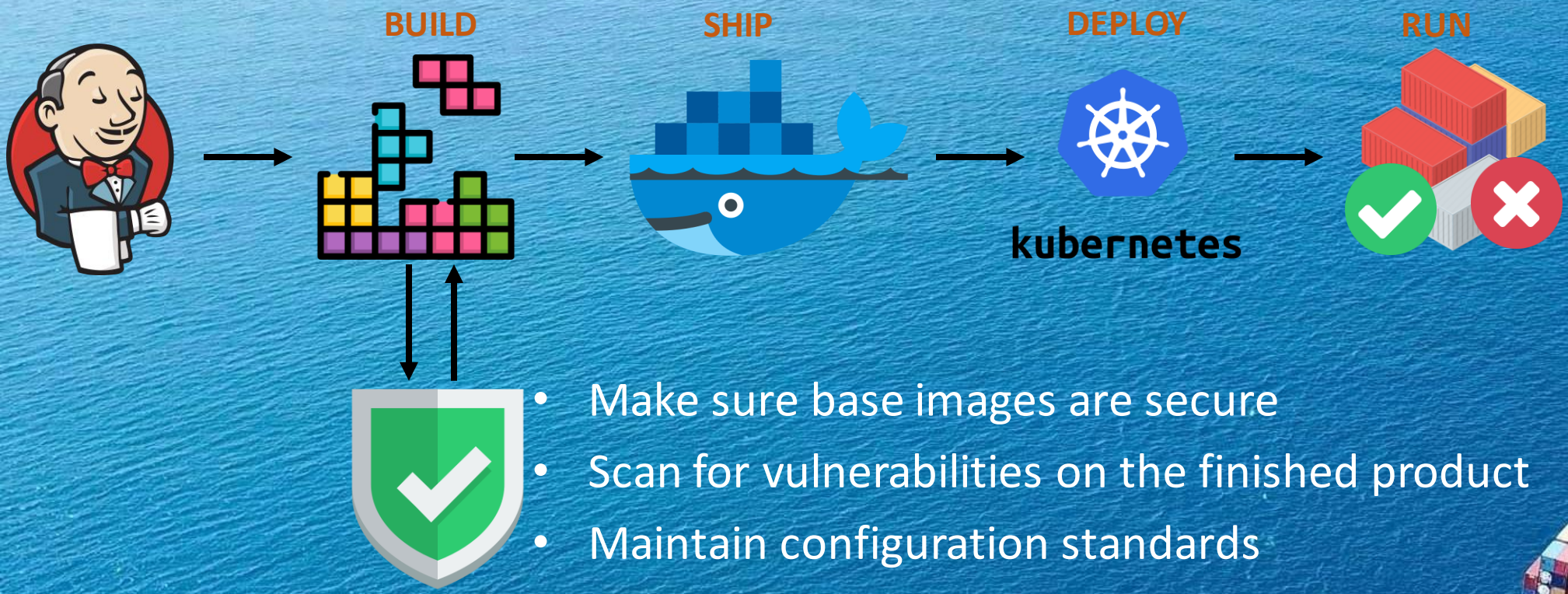



| Image Quality, Exposure Risk | Trust, Integrity Assurance | Resource, Access Control | Process, Network, Secrets |
|---|---|---|---|

VISIBILITY

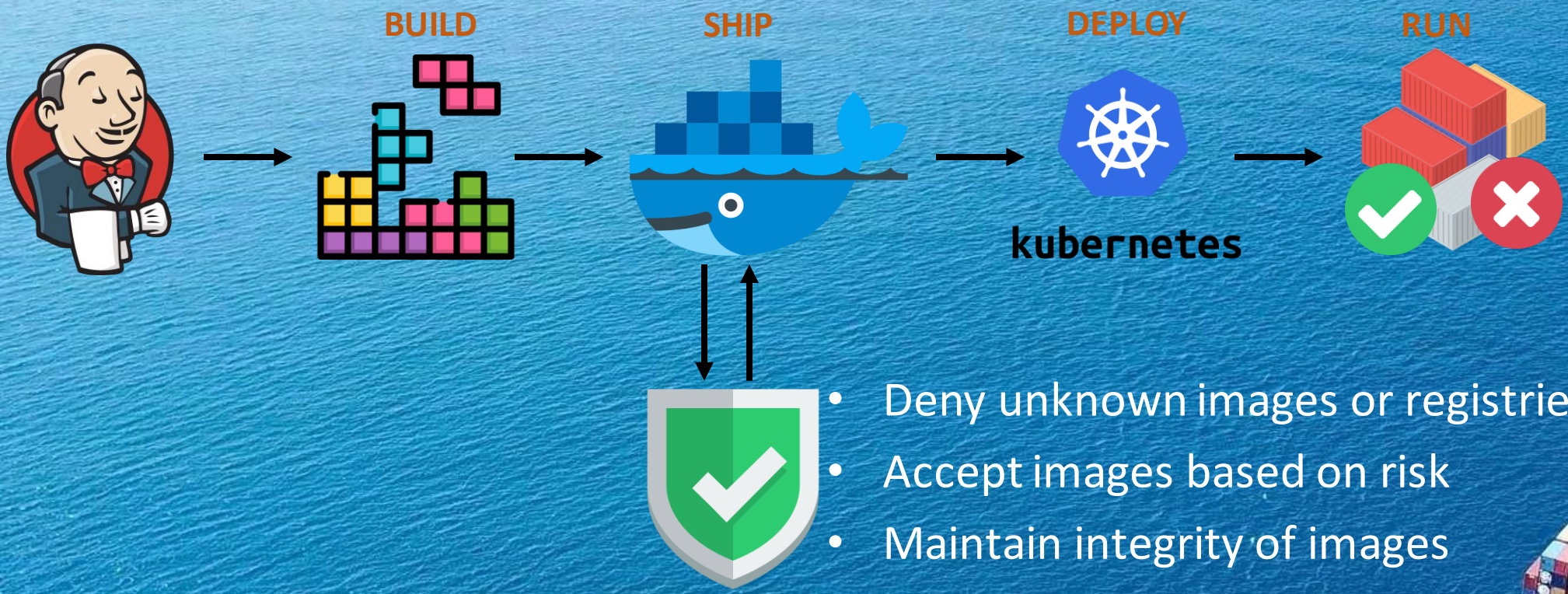
- Risk posture of the images is not completely understood
- Unclear as to where security should fit in the process
- Containers are not visible with current security tools
- Reliance on open source and external components used

# SECURITY STARTS IN THE BUNDLE PHASE



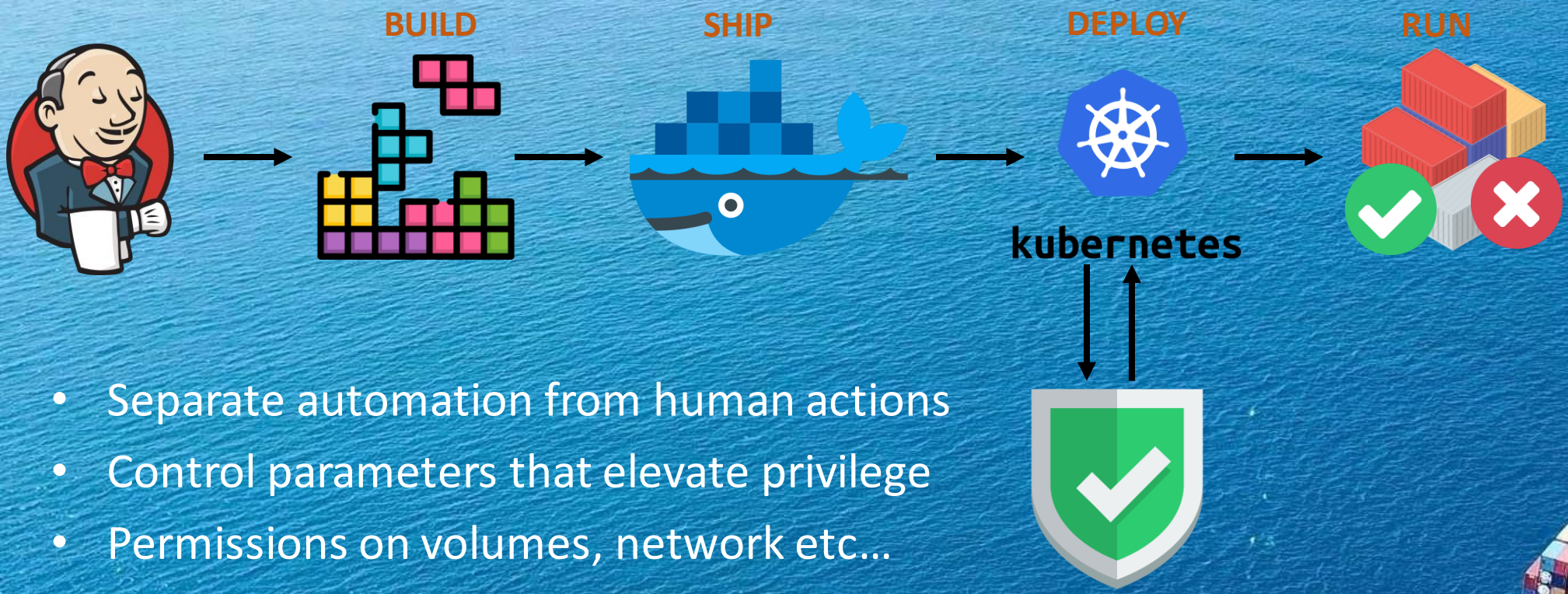

- Make sure base images are secure
- Scan for vulnerabilities on the finished product
- Maintain configuration standards
- Share info between Dev, Ops, Sec

# ADD ENFORCEMENT OF IMAGE USAGE



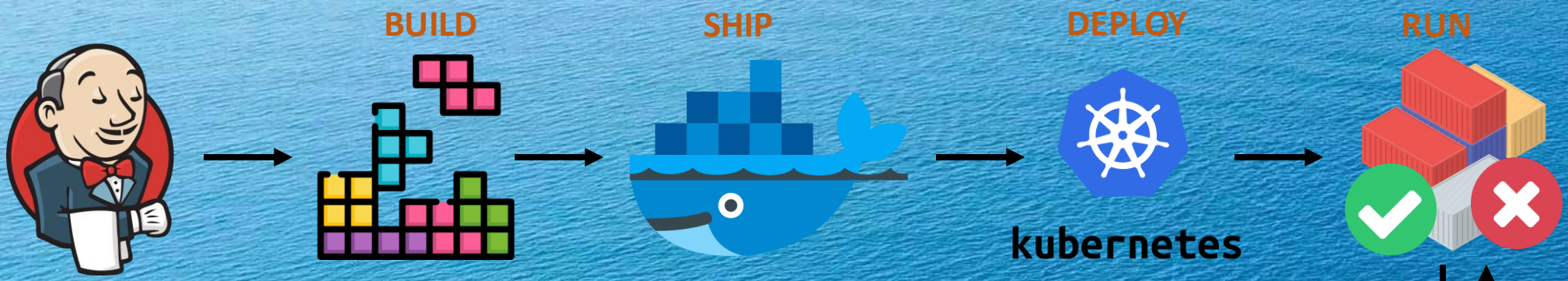

- Deny unknown images or registries
- Accept images based on risk
- Maintain integrity of images

# LIMIT ACCESS TO CONTAINER ENGINE




- Separate automation from human actions
- Control parameters that elevate privilege
- Permissions on volumes, network etc…
- Maintain audit trail with accountable user

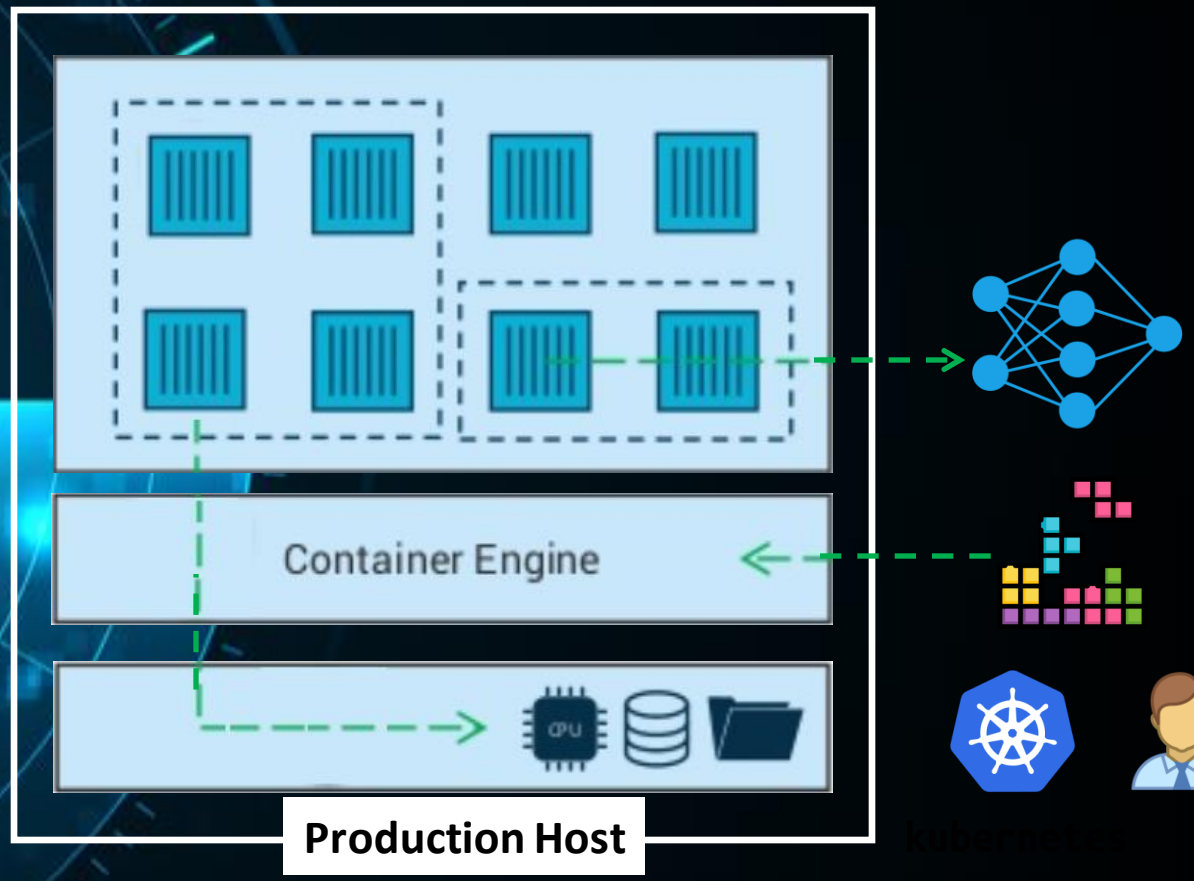
# GRANULAR RUNNING OF CONTAINERS



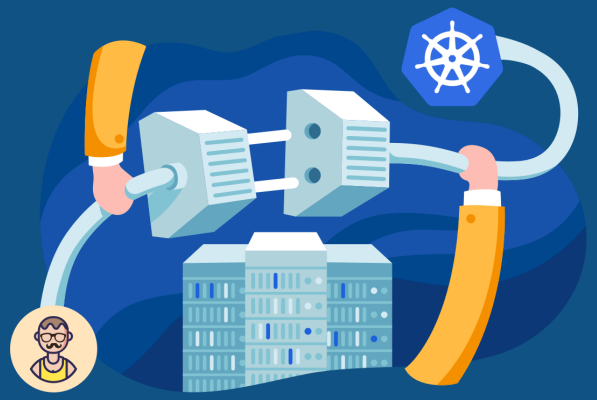

- Resource management
- Protect sensitive data
- Network segmentation
- Fine-grained app controls

# Let’s summarize container security

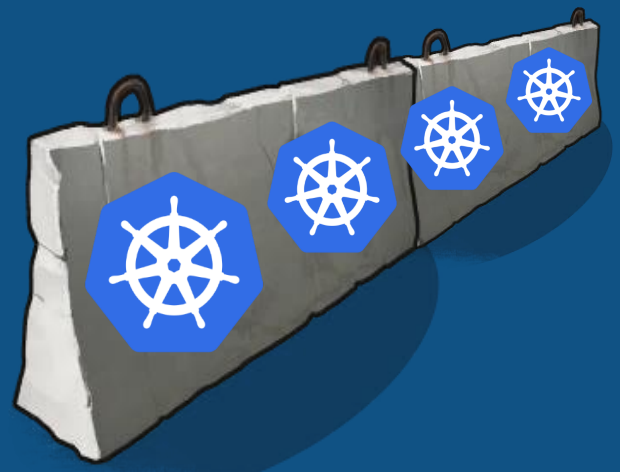
- Prevent unknown images
- Stop user privilege escalation
- Stop suspicious processes
- Enforce network isolation
- Encrypt sensitive vars/params
- Swear by automation and tools
- Visibility across the environment



# KUBERNETES SECURITY IMPROVEMENTS



Disable Anonymous Authentication



Configure Admission Controllers

Pod Security Policies

Enable Authz with RBAC

Network Policies for Segmentation

- **Disable anonymous authentication**

Generally two types of users exist:
– Normal Users (managed by outside entities like LDAP, AD etc.)
– Service Accounts (managed by K8s API with credentials as secrets)

Anything else is treated as anonymous request
– Given a username `system:anonymous` and group `system:unauthenticated`

For e.g. with token authentication & anonymous access enabled, a request with an invalid bearer token would receive a 401 error; a request providing no bearer token would be treated as an anonymous request.

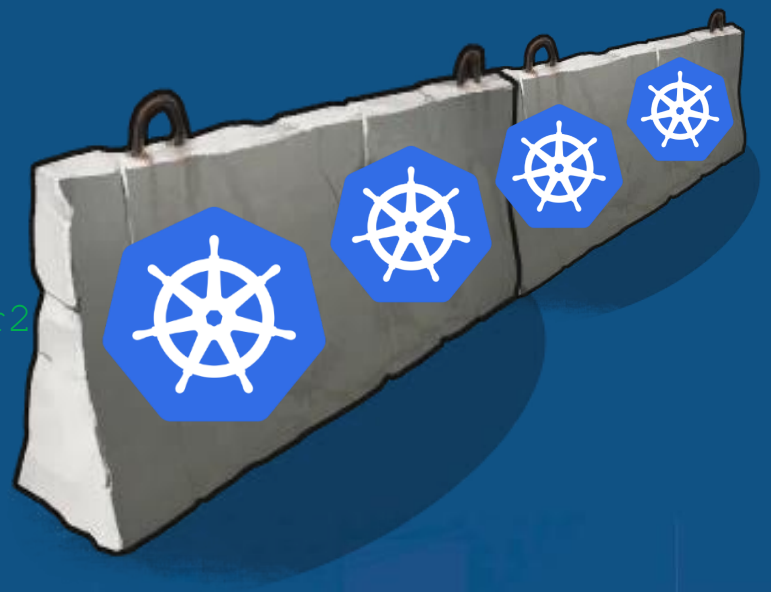
Recommendation:
Disable this mode by passing `--anonymous-auth=false` option to API server

- **Configure admission controllers**

  – Intercepts requests to create, delete, modify or connect to proxy
  – “validating” or “mutating” (able to modify the object they admit)

  Syntax: Replace what appears after = with name of the admission controller

```
--enable-admission-plugins=NameOfController,NameOfController2
```



Recommendation:

* Enable below admission controllers by default

```
LimitRanger,DefaultStorageClass,DefaultTolerationSeconds,Names
paceLifecycle,ServiceAccount,MutatingAdmissionWebhook,Validati
ngAdmissionWebhook,Priority,ResourceQuota,PodSecurityPolicy
```

* Enable `ValidatingAdmissionWebhook` to validate K8s resources during create, update, and delete operations

* Ensure `AlwaysPullImages` is set to make sure private images are only pulled by those who have the credentials.

- **Pod security policies**
  - Defined as a cluster level resource that controls security sensitive aspects
  - Conditions for pod to be admitted in system
  - Target pod's service account must be authorized for this policy
  - Implemented as an optional (but recommended) admission controller

- **Disable public access to cluster**
  - Never expose remote connectivity to your node\cluster
  - Use a bastion host in your management VPC
  - Ensure it is peered with the cluster network for connectivity

- **Enable authorization with RBAC**

  – Method of regulating access to resources based on the roles of individual users
  – RBAC uses the `rbac.authorization.k8s.io` API group to drive authz decisions

  Syntax: Start the API server with the `authorization-mode` flag for `RBAC`

```
kube-apiserver --authorization-mode=RBAC
```

Potential Mistakes:

– Make sure `cluster-admin` role is not granted unnecessarily (especially during the transition from legacy ABAC controller to RBAC)

– Role aggregation (K8s v1.9), if not carefully reviewed can lead to improper usage

– Duplicated role grant may happen; subjects get same access in more than one way

– Unused roles to subjects that do not exist (deleted service accounts) can make it difficult to see configurations that *do* matter.

- **Network policies for segmentation**

  – Specification of how groups of pods can communicate with each other and other network endpoints

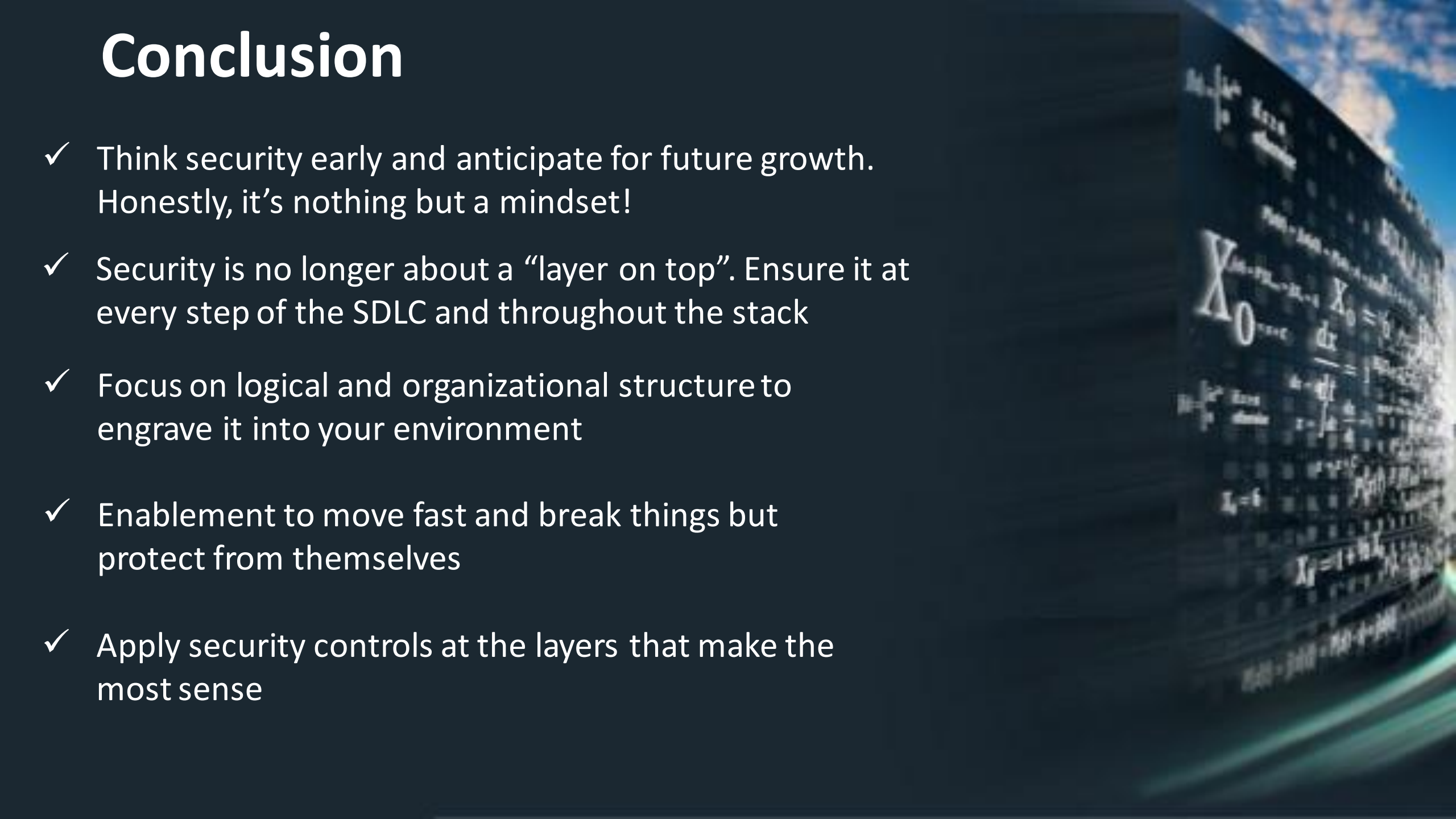
  – When no network policies are applied, then all connections to and from it are permitted

  – Notion of “pod isolation” which means that pods are isolated if at least one network policy applies to them

  – Intricate and difficult to understand for proper usage. Network plugins recommended from Calico, Cilium, Kube-router, Romana and Weave Net

# Conclusion

- ✓ Think security early and anticipate for future growth. Honestly, it's nothing but a mindset!
- ✓ Security is no longer about a "layer on top". Ensure it at every step of the SDLC and throughout the stack
- ✓ Focus on logical and organizational structure to engrave it into your environment
- ✓ Enablement to move fast and break things but protect from themselves
- ✓ Apply security controls at the layers that make the most sense



## Questions | Comments | Discussions

## Thanks for listening.


**Runcy Oommen**

runcyoommen

https://runcy.me