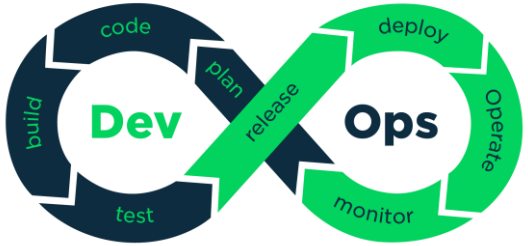
DevSecOps



Raman Khanna

Introduction

Name

Background – Development / Infrastructure / Database / Network

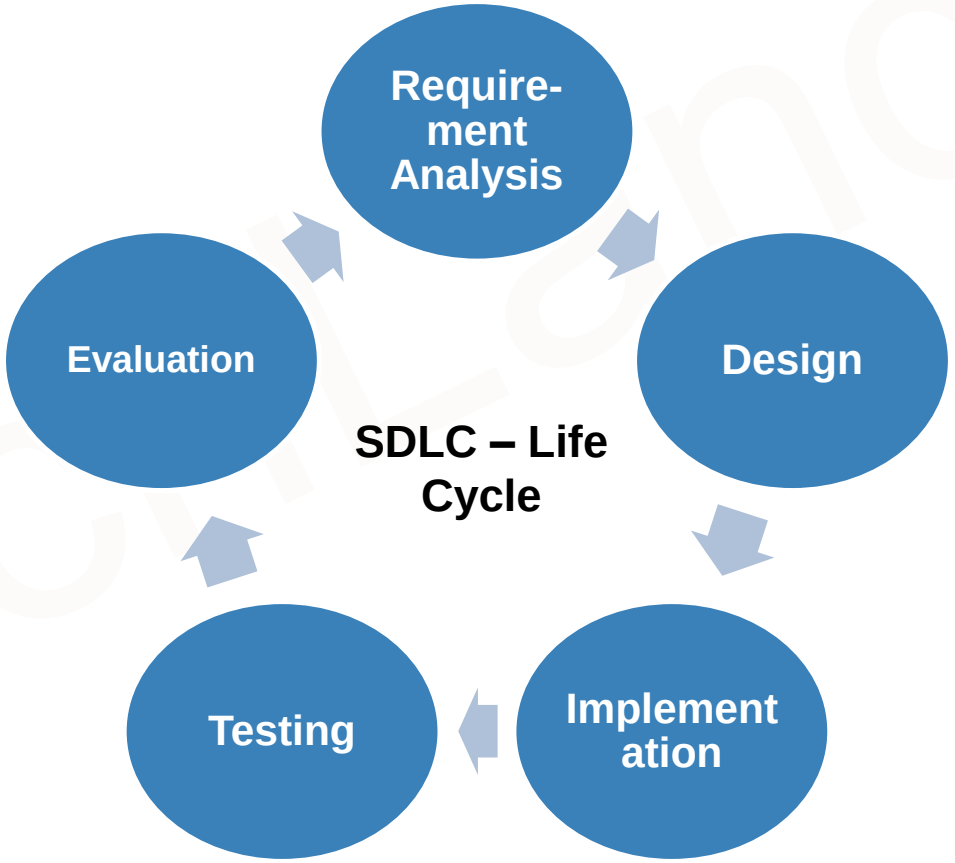
Experience : AWS/Git/Docker/Kubernetes/Jenkins/Terraform

DevOps

What is DevOps?

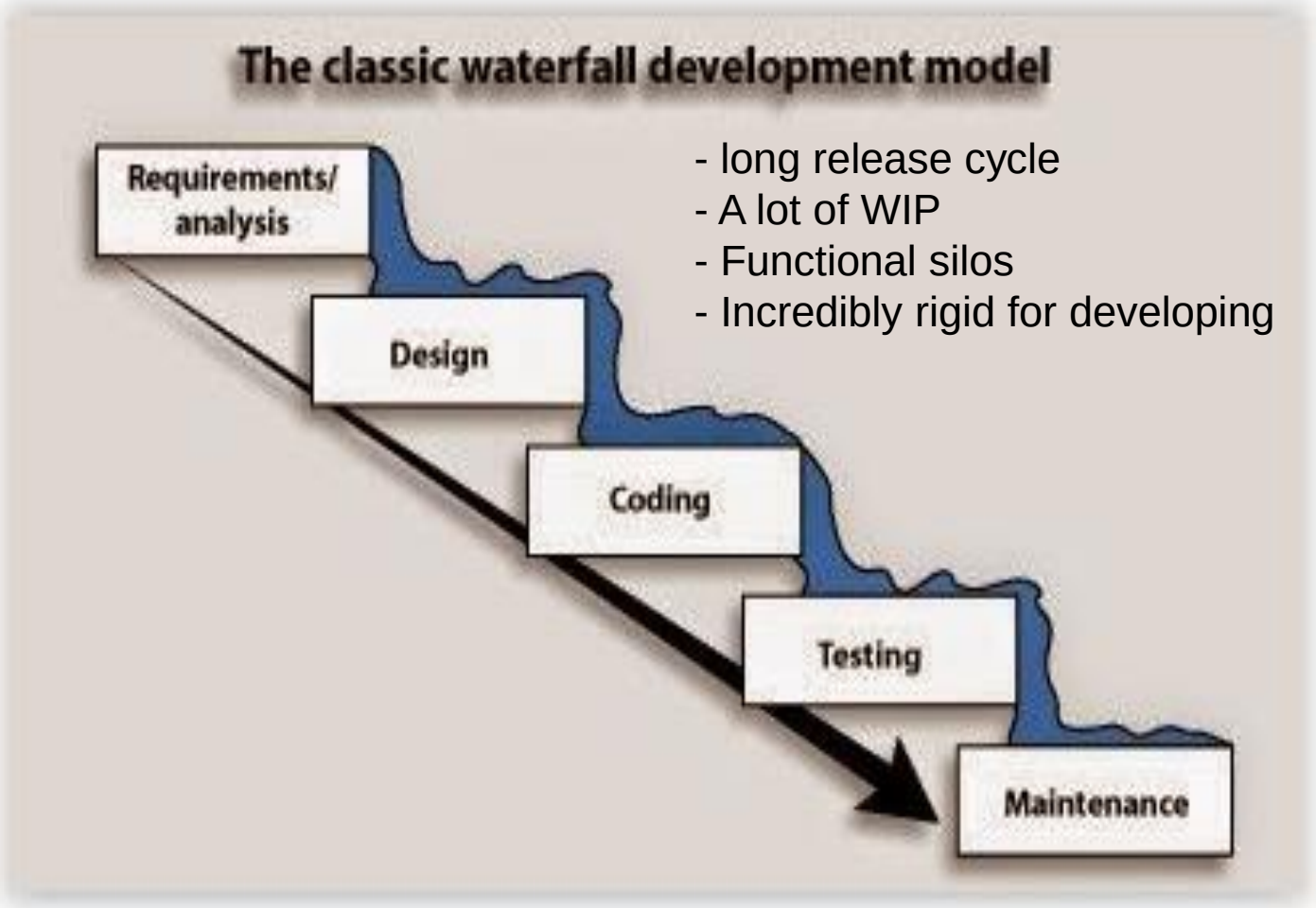
SDLC Model

- A systems development life cycle is composed of **several clearly defined and distinct work phases** which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems

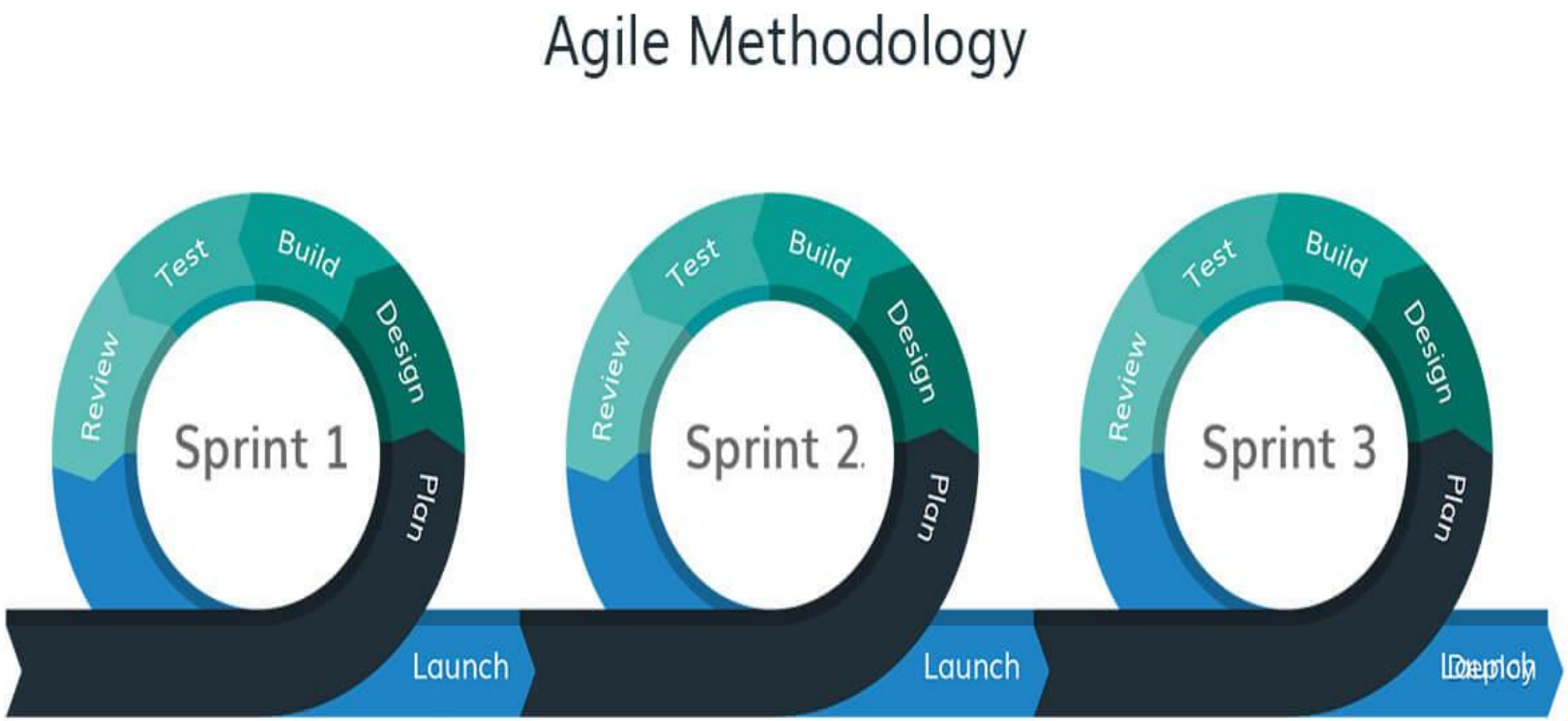


Waterfall Model

1. Determine the Requirements
2. Complete the design
3. Do the coding and testing (unit tests)
4. Perform other tests (functional tests, non-functional tests, Performance testing, bug fixes etc.)
5. At last deploy and maintain

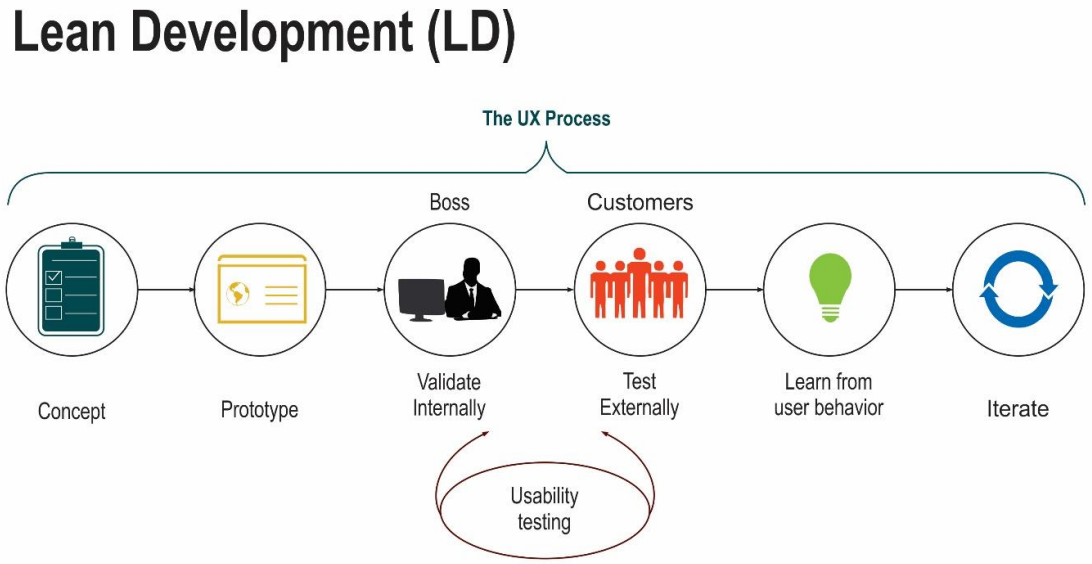


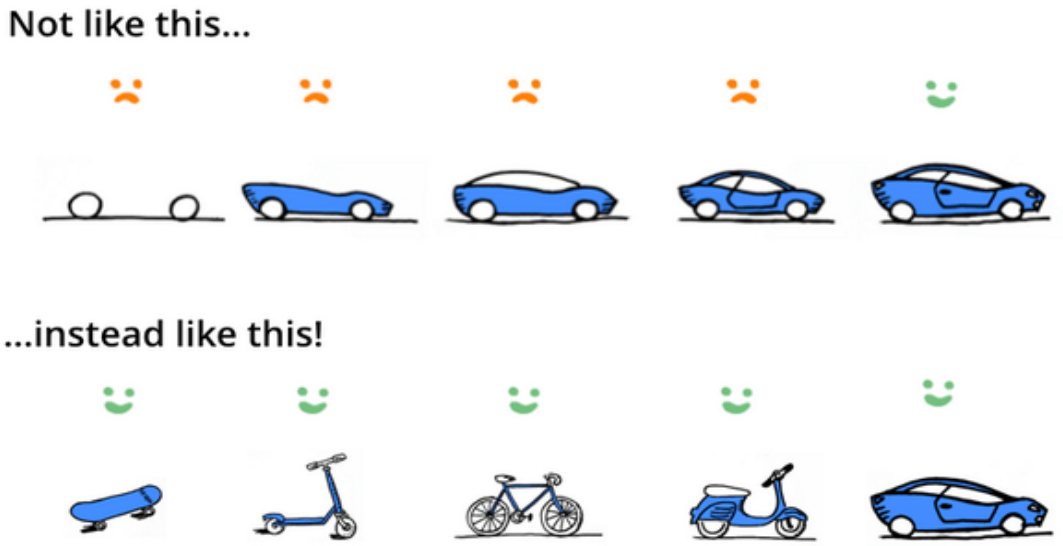
Agile



- Shorter release cycle
- Small batch sizes (MVP)
- Cross-functional teams
- Incredibly agile

Lean Development

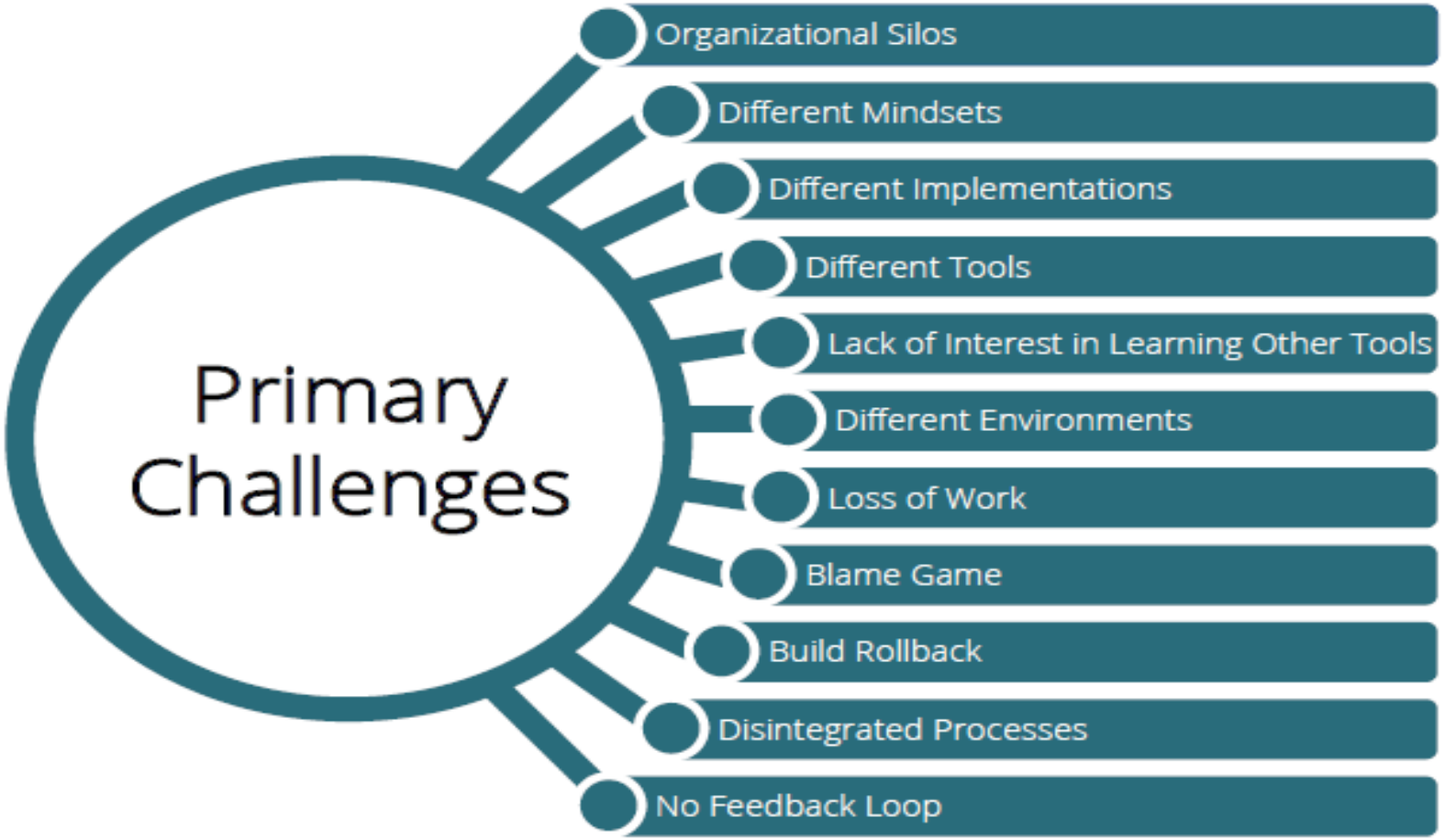




- Suddenly ops was the bottleneck (more release less people), again WIP is more!

Challenges

Some of the challenges with the traditional teams of Development and Operations are:



A Typical Case Study

Development Team:

- Monday Morning, the writing of code done, unit tests completed, code delivered to the Integration teams to get the code included in CI builds.
- To get the services tested, a ticket is opened for QA teams

Build/Release/Testing/Integration Team:

- Tuesday Morning, ticket accepted, a tester put an email to the developer asking deployment instructions. There is not automated deployments, developer updated to the tester, lets come online and we will deploy the services to the QA environment together.
- Call started, developer identified the “test environment” is not compatible.
- Tuesday afternoon, a ticket raised in Ops Team with new specifications.

Ops Team:

- Wednesday morning, ticket accepted, specifications checked , a new port open request was identified.
- Ticket raised for Security team, ticket accepted, change approved, port opened, email received by the Ops team the work is done.

A Typical Case Study

Ops Team:

- Identified the provisioning requirements again and started work on building the environment.

Build/Release/Testing/Integration Team:

- Thursday Morning, updates received - the environment is ready. Developer and Tester again on call to deploy new services. Services deployed; tester is running test scripts. Next phase is to run regression test cases. Again a new ticket is raised for new test data with production teams and day ends.

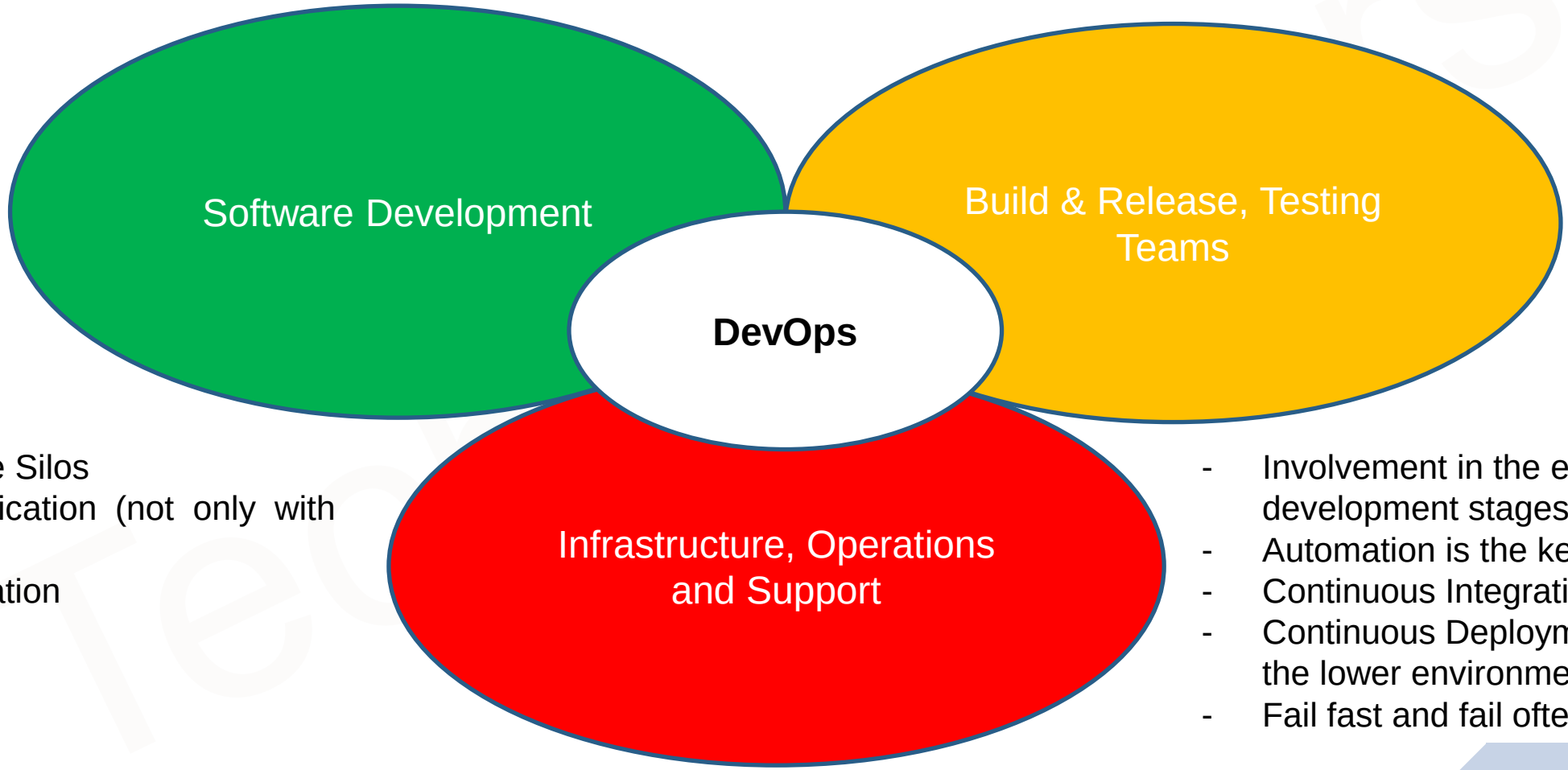
Ops Team:

- Its Friday and the work is not on full swing, ticket accepted but not worked as production team has to complete rest of the works. Somehow the test data is gathered by Friday Evening.

Build/Release/Testing/Integration Team:

- Monday morning, tester gets the data, regression tests run, a defect found, and ticket returned to the development team.

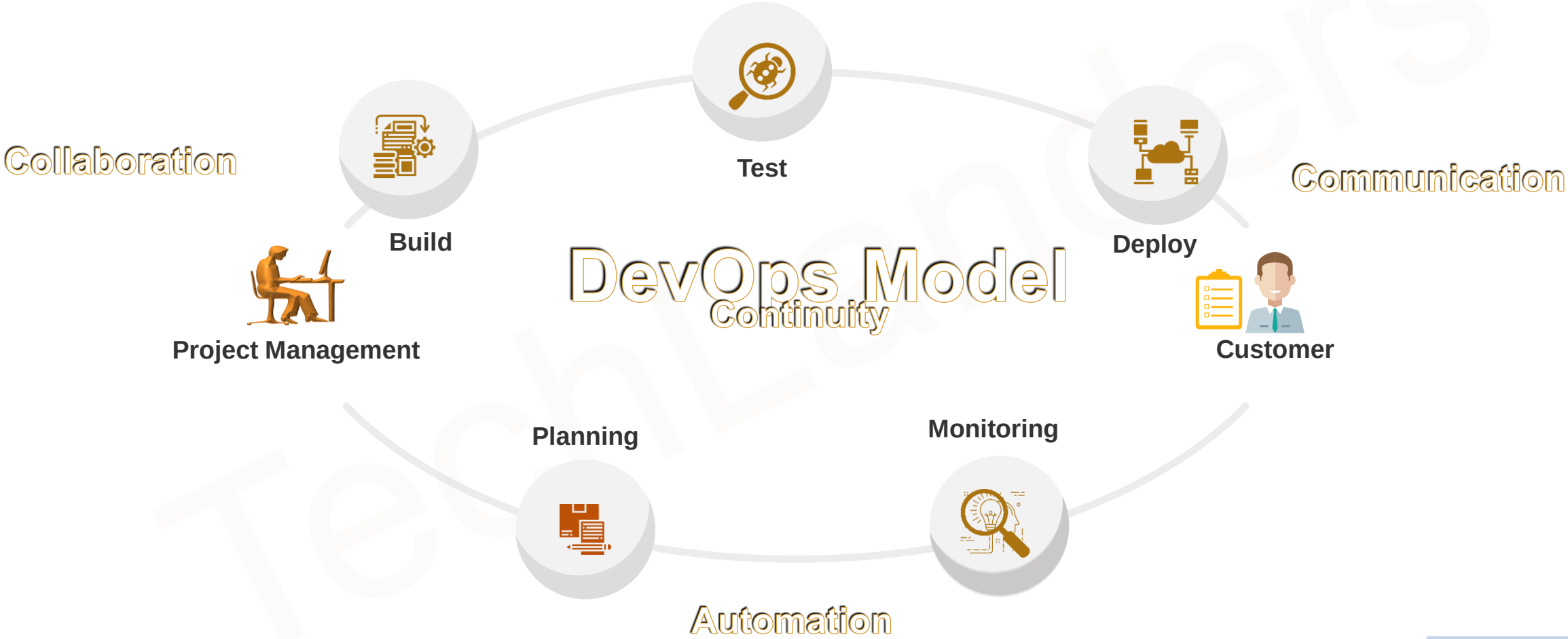
DevOps



- Break the Silos
- Communication (not only with emails)
- Collaboration
- Trust

- Involvement in the early development stages
- Automation is the key
- Continuous Integration
- Continuous Deployments in the lower environments
- Fail fast and fail often

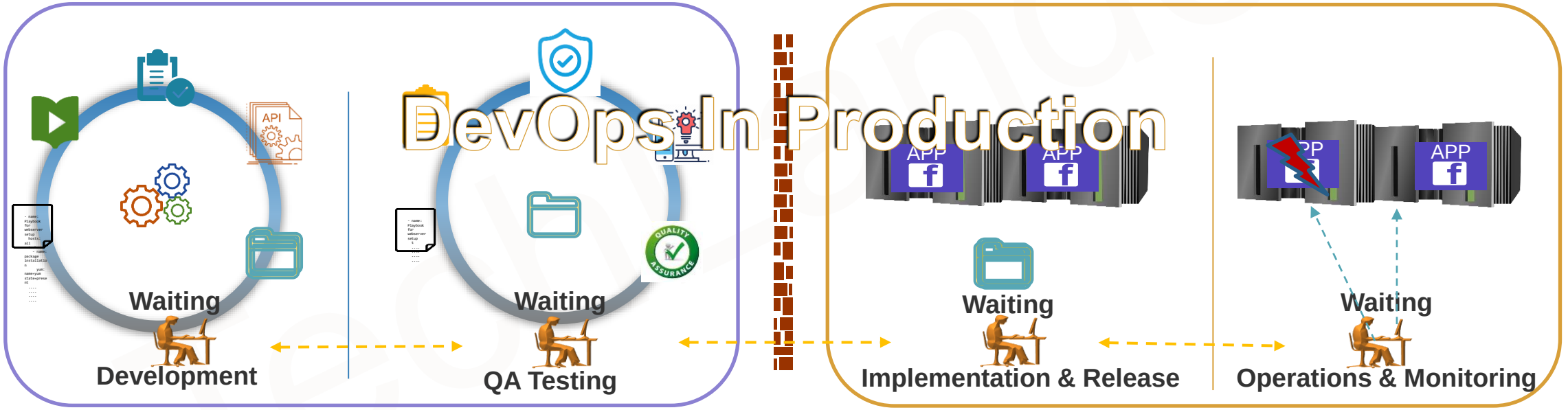
DevOps



DevOps in Action

Continuous Feedback Continuous Improvement Continuous Planning

Continuous Integration Continuous Delivery Continuous Deployment Continuous Monitoring



DevOps Essence

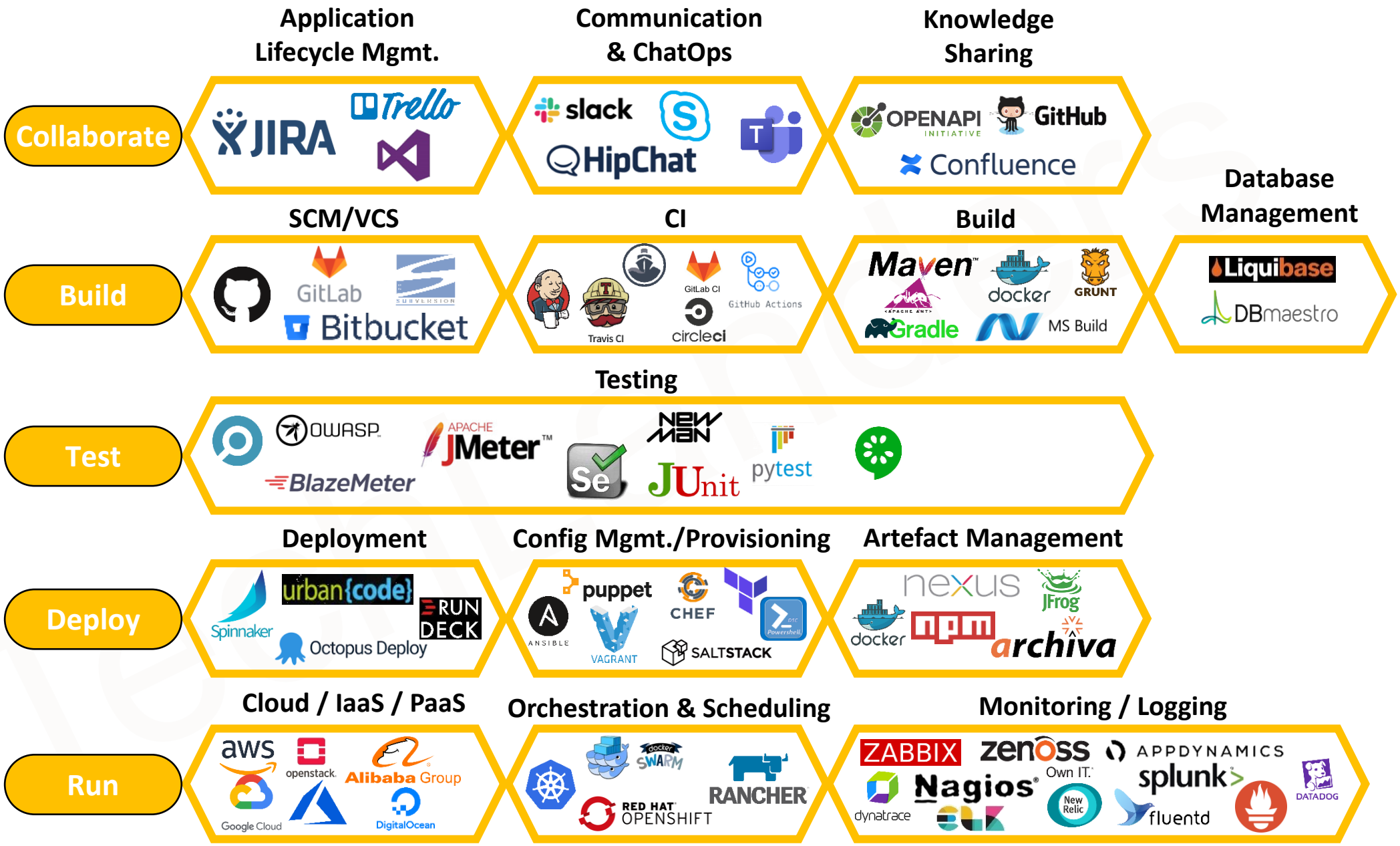
Efficiency - Faster time to market

Predictability - Lower failure rate of new releases

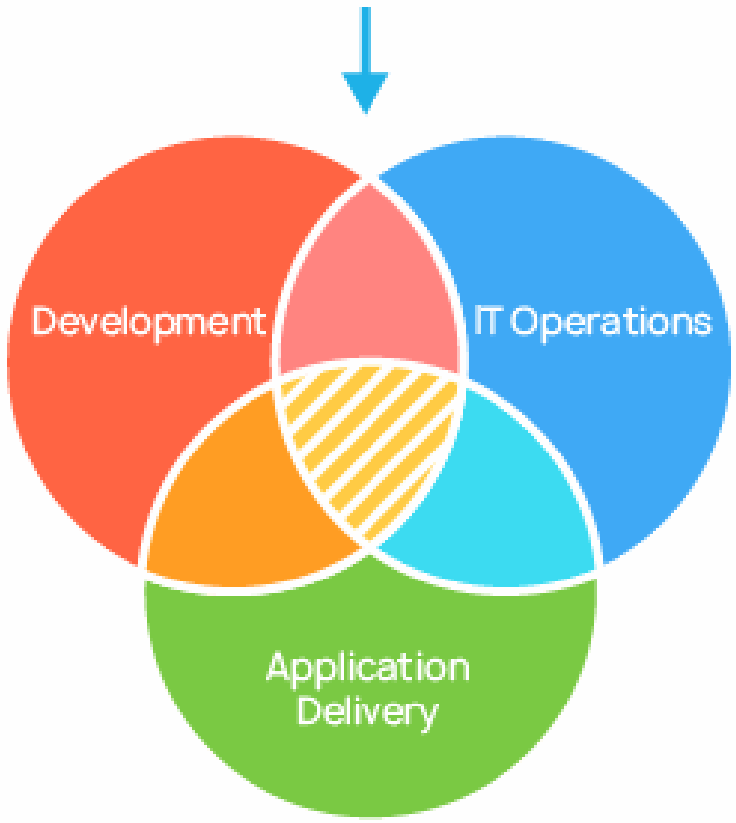
Reproducibility – Version everything

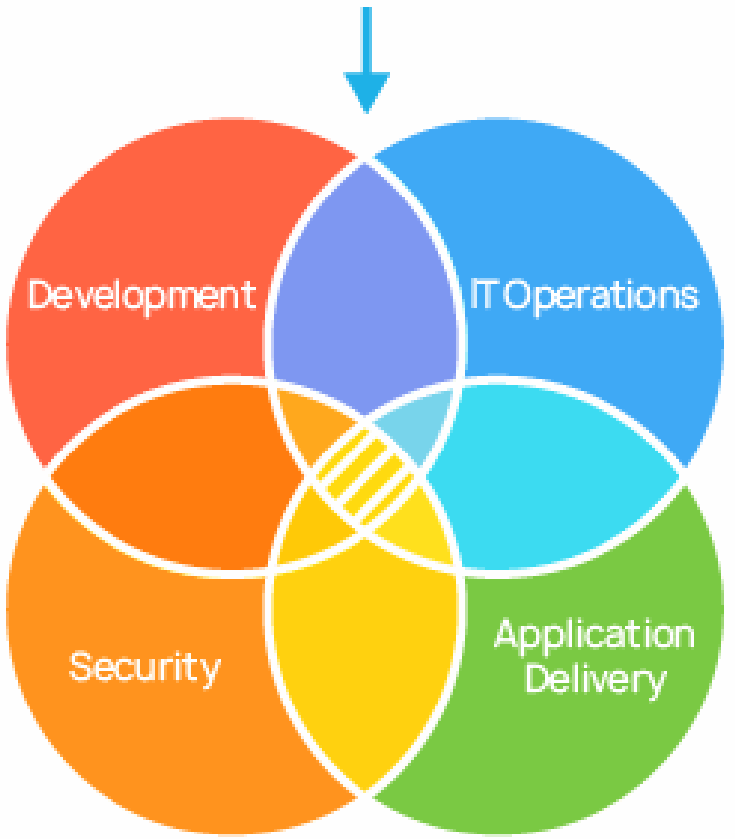
Maintainability - Faster time to recovery in the event of a new release crashing or otherwise disabling the current system

DevOps Toolsets

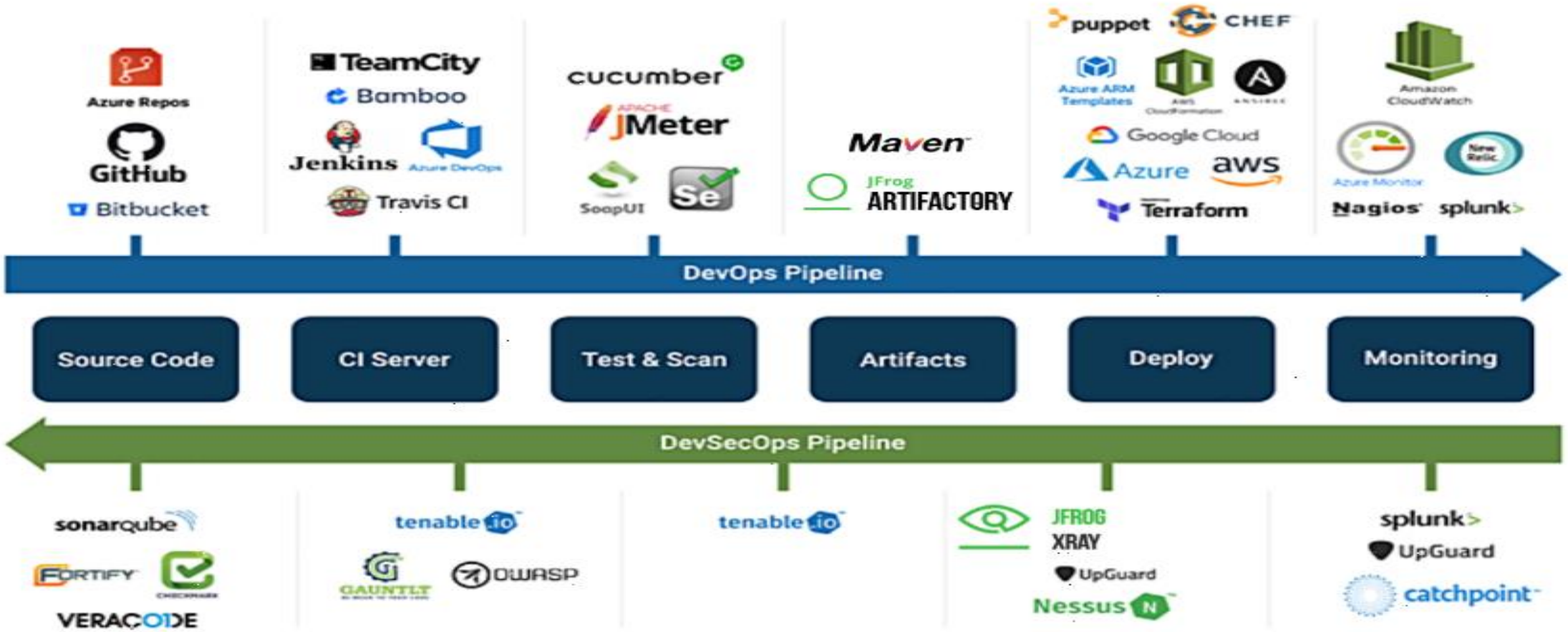


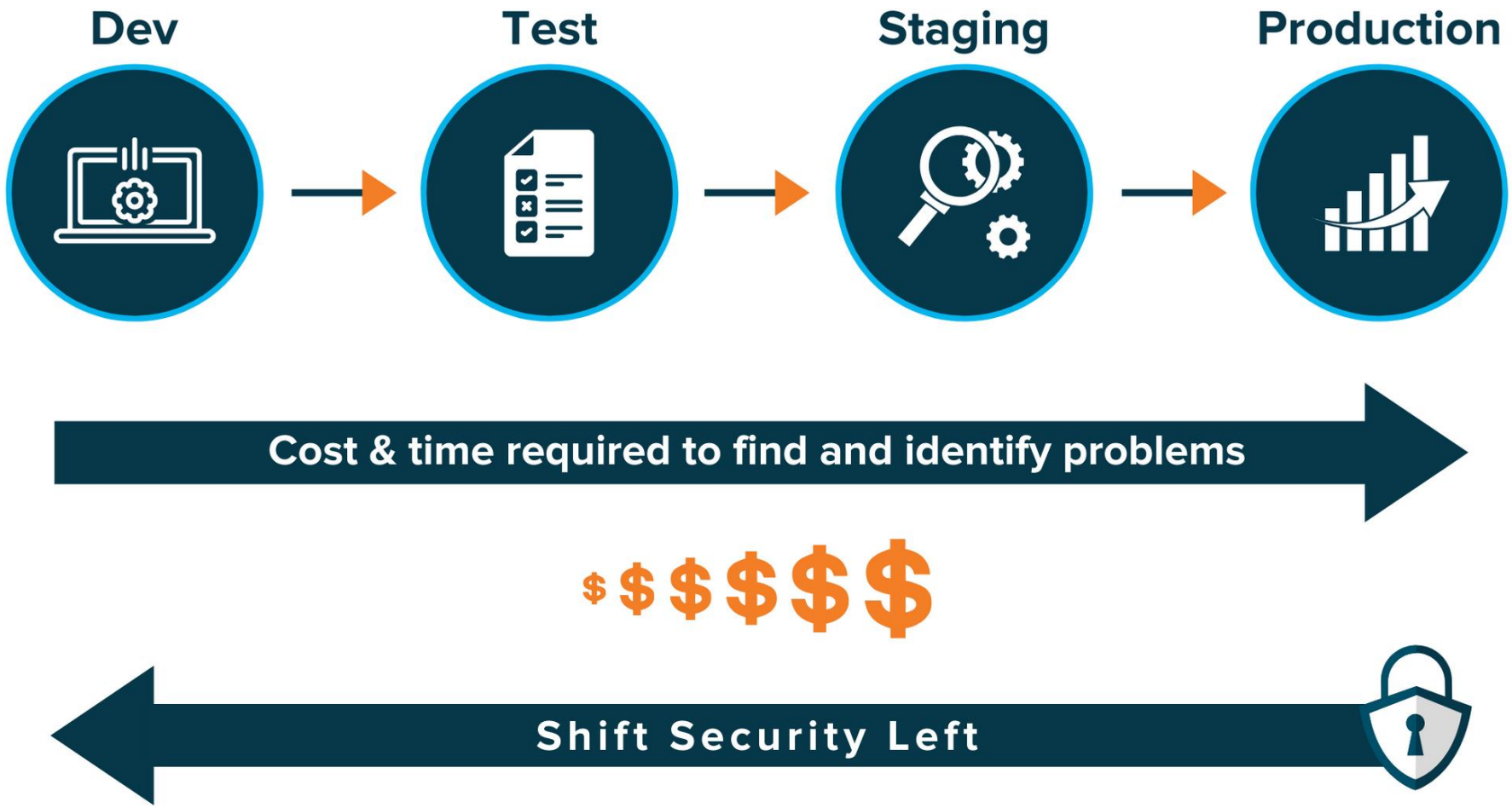
DevOps VS DevSecOps





DevOps


Azure Repos
GitHub
Bitbucket
TeamCity
Bamboo
Jenkins
Azure DevOps
Travis CI
cucumber
JMeter
SoapUI
Maven
JFrog
ARTIFACTORY
puppet
CHEF
Azure ARM Templates
Google Cloud
Azure
aws
Terraform
Amazon CloudWatch
Azure Monitor
Nagios
splunk>
DevOps Pipeline
Source Code
CI Server
Test & Scan
Artifacts
Deploy
Monitoring
DevSecOps Pipeline
sonarqube
FORTIFY
VERACODE
tenable.io
GAUNTLT
OWASP
tenable.io
JFROG
XRAY
UpGuard
Nessus
splunk>
UpGuard
catchpoint


Dev
Test
Staging
Production
Cost & time required to find and identify problems
$ $ $ $ $ $
Shift Security Left

NVD, CVE, and CVSS

- NVD == National Vulnerability Database
  - SCAP == Security Content Automation Protocol
- CVE == Common Vulnerabilities and Exposure
- CVSS == Common Vulnerability Scoring System

1. **NVD (National Vulnerability Database)**:
    1. The NVD is a repository of standards-based vulnerability management data represented using the Security Content Automation Protocol (SCAP).
    2. It provides information about vulnerabilities, including their severity, affected products, and fixes.
    3. NVD is curated by the National Institute of Standards and Technology (NIST) in the United States.
2. **CVE (Common Vulnerabilities and Exposures)**:
    1. CVE is a dictionary of common identifiers for publicly known cybersecurity vulnerabilities.
    2. Each CVE entry includes a unique identifier (CVE ID) along with a description of the vulnerability.
    3. CVE IDs are used as references in security advisories, vulnerability databases, and security tools.
    4. The CVE List is maintained by the MITRE Corporation, a not-for-profit organization that operates various research and development centers funded by the U.S. government.
3. **CPE (Common Platform Enumeration)**:
    1. CPE is a structured naming scheme for information technology systems, platforms, and packages.
    2. It provides a standardized format to describe and identify hardware, software, and other IT assets.
    3. CPE entries include vendor names, product names, version numbers, and other attributes to uniquely identify specific configurations or instances of software/hardware.
    4. CPE is used in vulnerability databases and scanning tools to associate vulnerabilities with specific IT assets.

NVD

- https://nvd.nist.gov/
- Allows searching of Vulnerability Database
  - https://nvd.nist.gov/vuln/search
- Uses SCAP
- Reports “publicly known” vulnerabilities
- Some criticism that it lags actual vulnerability discovery

CVE

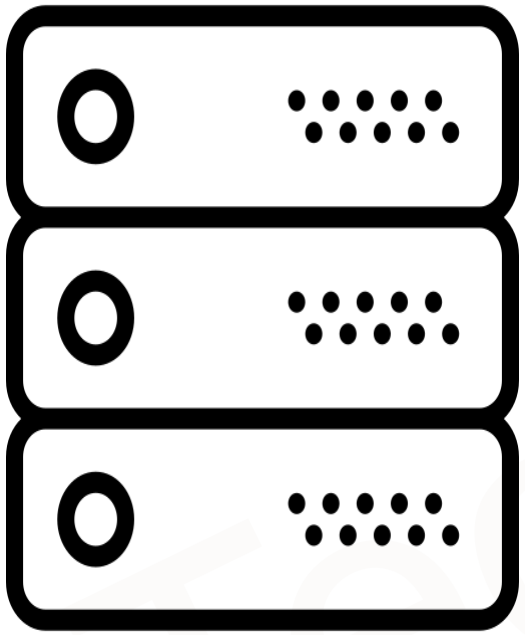
- https://cve.mitre.org/
- Is a listing for publicly known vulnerabilities

What are the Differences?

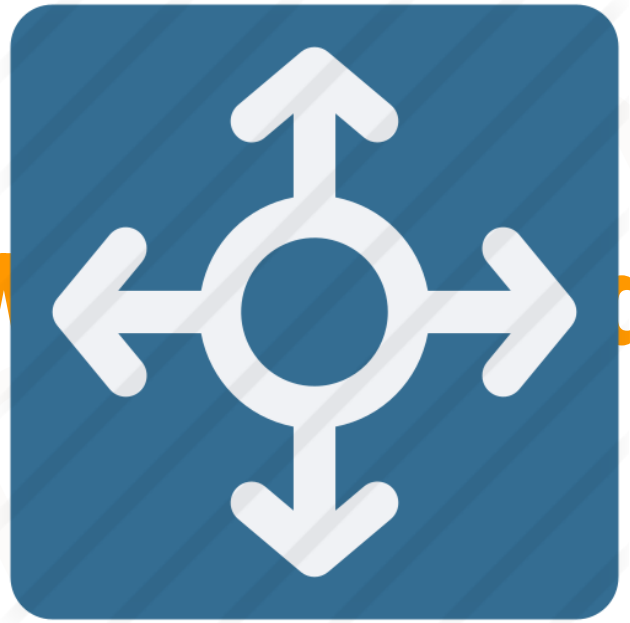
- NVD and CVE:
  - CVE is a part of NVD.
  - NVD allows more analysis and search capabilities
  - NVD and CVE are synchronized.
- CVE and CVSS
  - CVE is a free and open source standard and does not provide severity scoring as does CVSS

Cloud Fundamentals

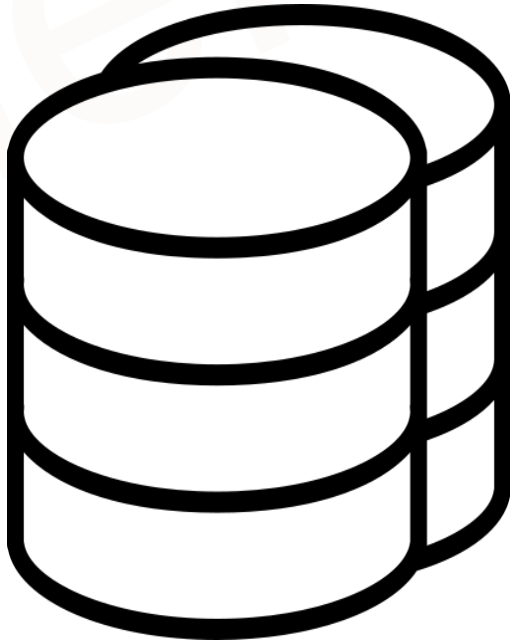
Traditional Datacenters



Servers

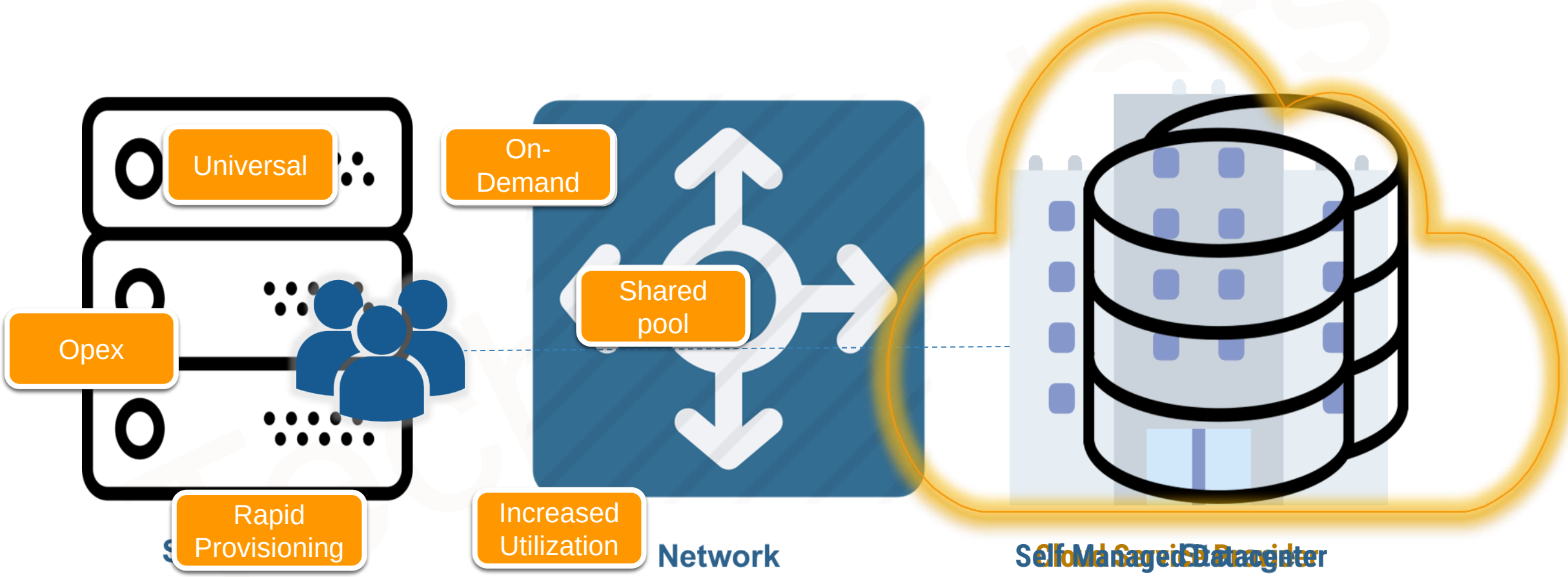


Network

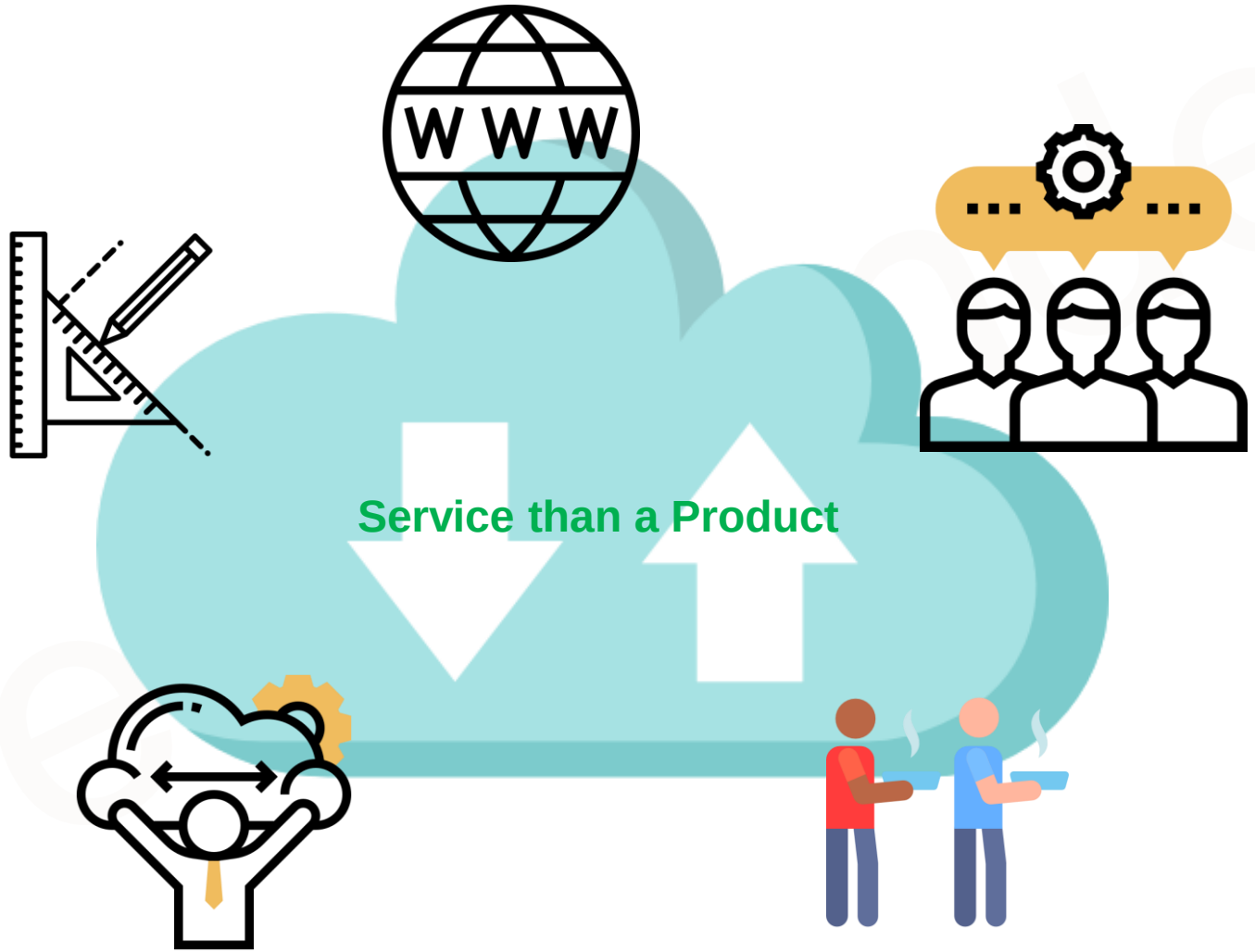


Storage

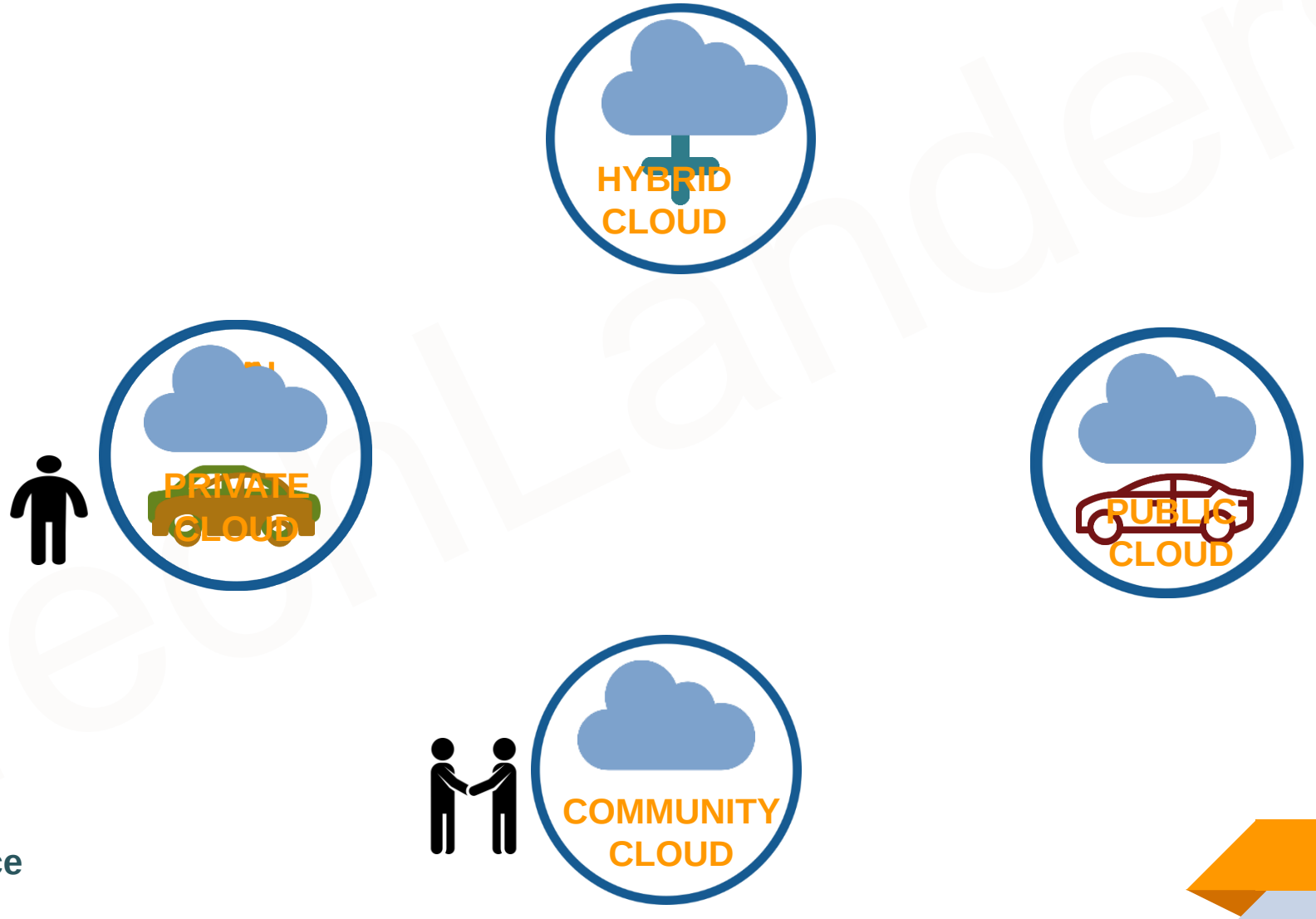
Traditional vs Cloud Datacenters



Characteristics of Cloud

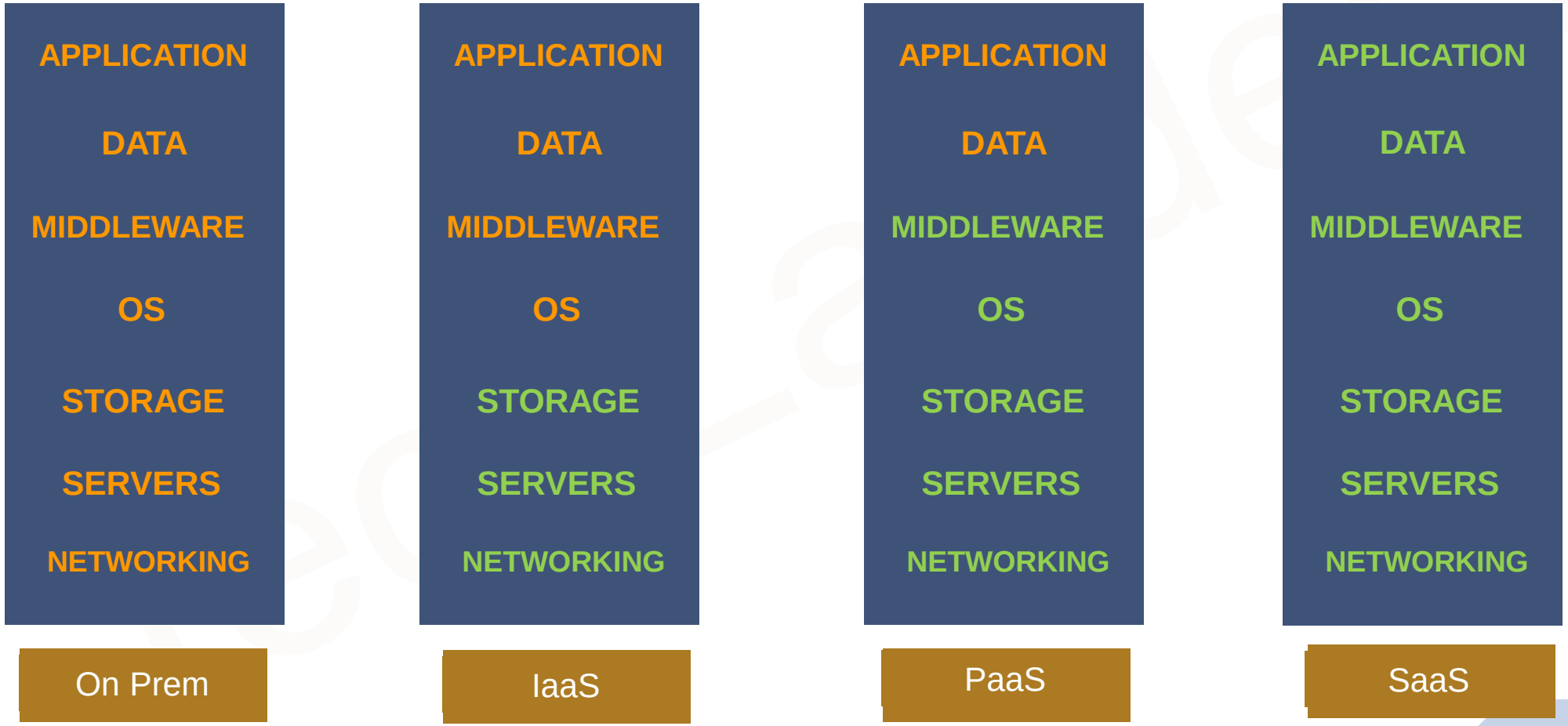


Cloud Deployment Types



Service Models - CARaaS

Service Models



Cloud Benefits

Cloud Major Use Cases

On and Off

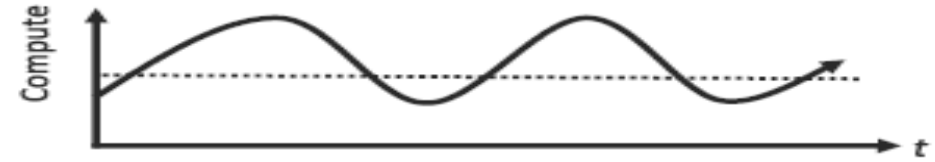
On and off workloads (e.g. batch job)
Over provisioned capacity is wasted
Time to market can be cumbersome

Growing Fast

Successful services needs to grow/scale
Keeping up with growth is a big IT challenge
Cannot provision hardware fast enough

Unpredictable Bursting

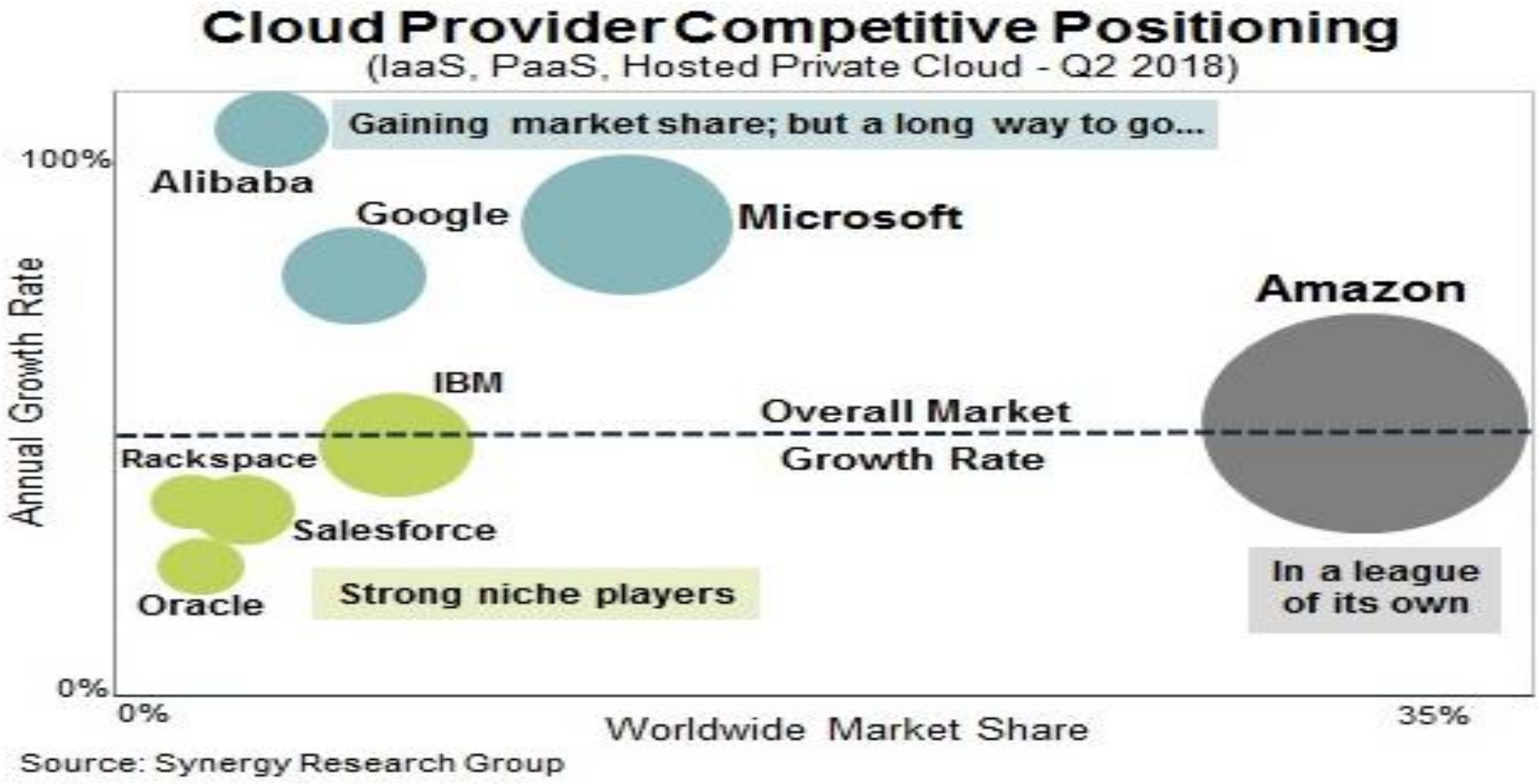
Unexpected/unplanned peak in demand
Sudden spike impacts performance
Cannot over provision for extreme cases



Predictable Bursting

Services with micro seasonality trends
Peaks due to periodic increased demand
IT complexity and wasted capacity

Cloud Players



AWS

Amazon Web Services

AWS (Amazon Web Services) is a group of web services (also known as cloud services) being provided by Amazon since 2006.

AWS provides huge list of services starting from basic IT infrastructure like CPU, Storage as a service, to advance services like Database as a service, Serverless applications, IOT, Machine Learning services etc..

Hundreds of instances can be build and use in few minutes as and when required, which saves ample amount of hardware cost for any organizations and make them efficient to focus on their core business areas.

Currently AWS is present and providing cloud services in more than 190 countries.

Well-known for IaaS, but now growing fast in PaaS and SaaS.

Why AWS?

Low Cost: AWS offers, pay as you go pricing. AWS models are usually cheapest among other service providers in the market.

Instant Elasticity: You need 1 server or 1000's of servers, AWS has a massive infrastructure at backend to serve almost any kind of infrastructure demands, with pay for what you use policy.

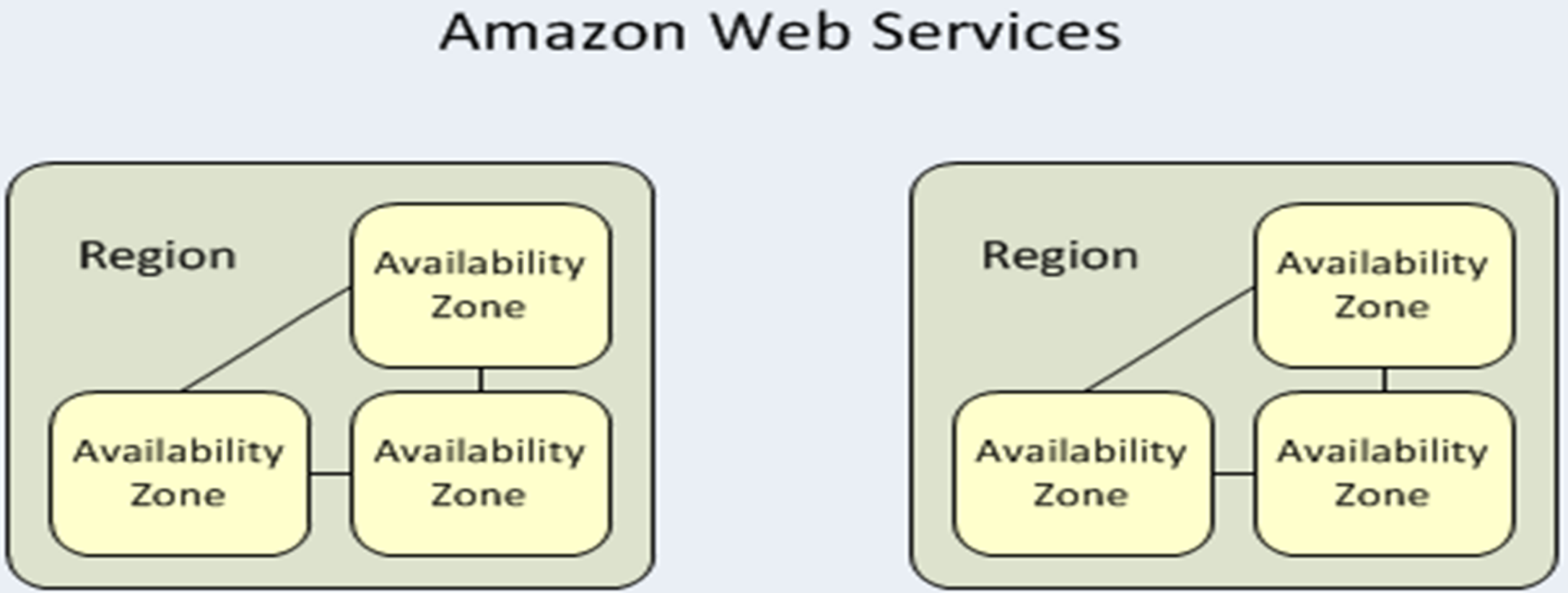
Scalability: Facing some resource issues, no problem within seconds you can scale up the resources and improve your application performance. This cannot be compared with traditional IT datacenters.

Multiple OS's: Choice and use any supported Operating systems.

Multiple Storage Options: Choice of high I/O storage, low cost storage. All is available in AWS, use and pay what you want to use with almost any scalability.

Secure: AWS is PCI DSS Level1, ISO 27001, FISMA Moderate, HIPAA, SAS 70 Type II passed. In-fact systems based on AWS are usually more secure than in-house IT infrastructure systems.

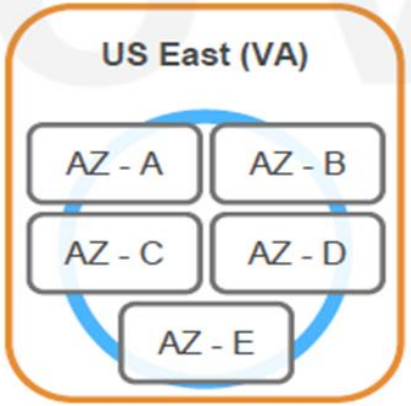
Amazon Web Services

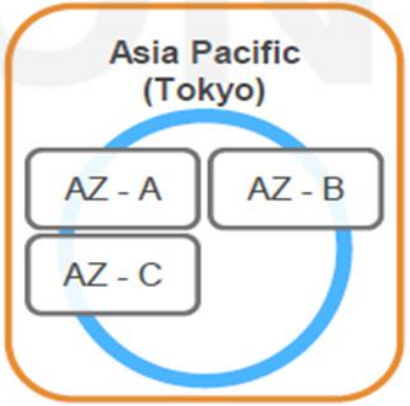


Amazon Web Services

At least 2 AZs per region.

- Examples:
  - US East (N. Virginia)
    - us-east-1a
    - us-east-1b
    - us-east-1c
    - us-east-1d
    - us-east-1e
  - Asia Pacific (Tokyo)
    - ap-northeast-1a
    - ap-northeast-1b
    - ap-northeast-1c





Note: Conceptual drawing only. The number of Availability Zones (AZ) may vary.

Amazon Web Services

AWS Regions:
-Geographic Locations
-Consists of at least two Availability Zones(AZs)
-All of the regions are completely independent of each other with separate Power Sources, Cooling and Internet connectivity.

AWS Availability Zones
-AZ is a distinct location within a region
-Each Availability Zone is isolated, but the Availability Zones in a Region are connected through low-latency links.
-Each Region has minimum two AZ’s
-Most of the services/resources are replicated across AZs for HA/DR purpose.

Amazon Web Services

Current:

22 AWS Regions

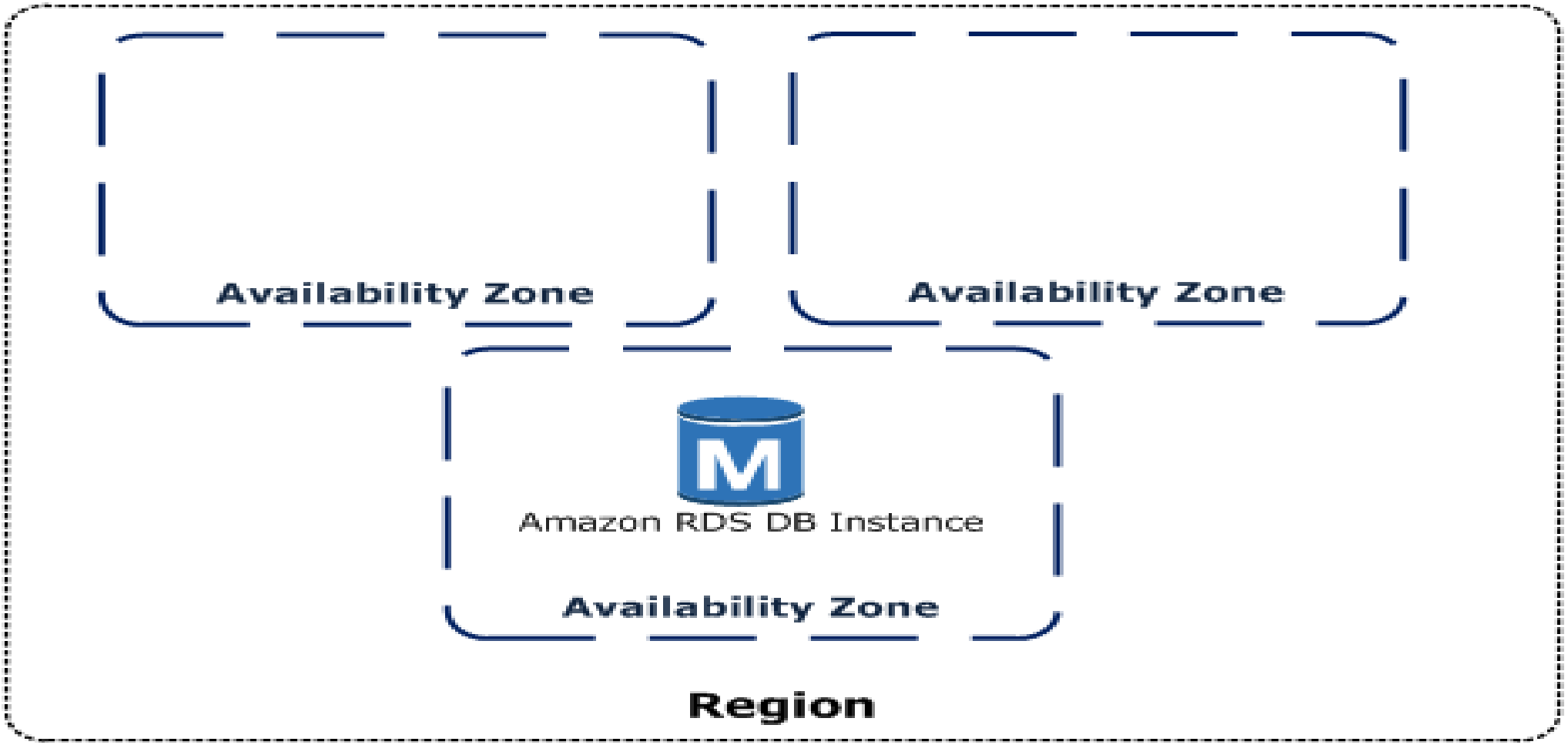
69 AZs

Upcoming:

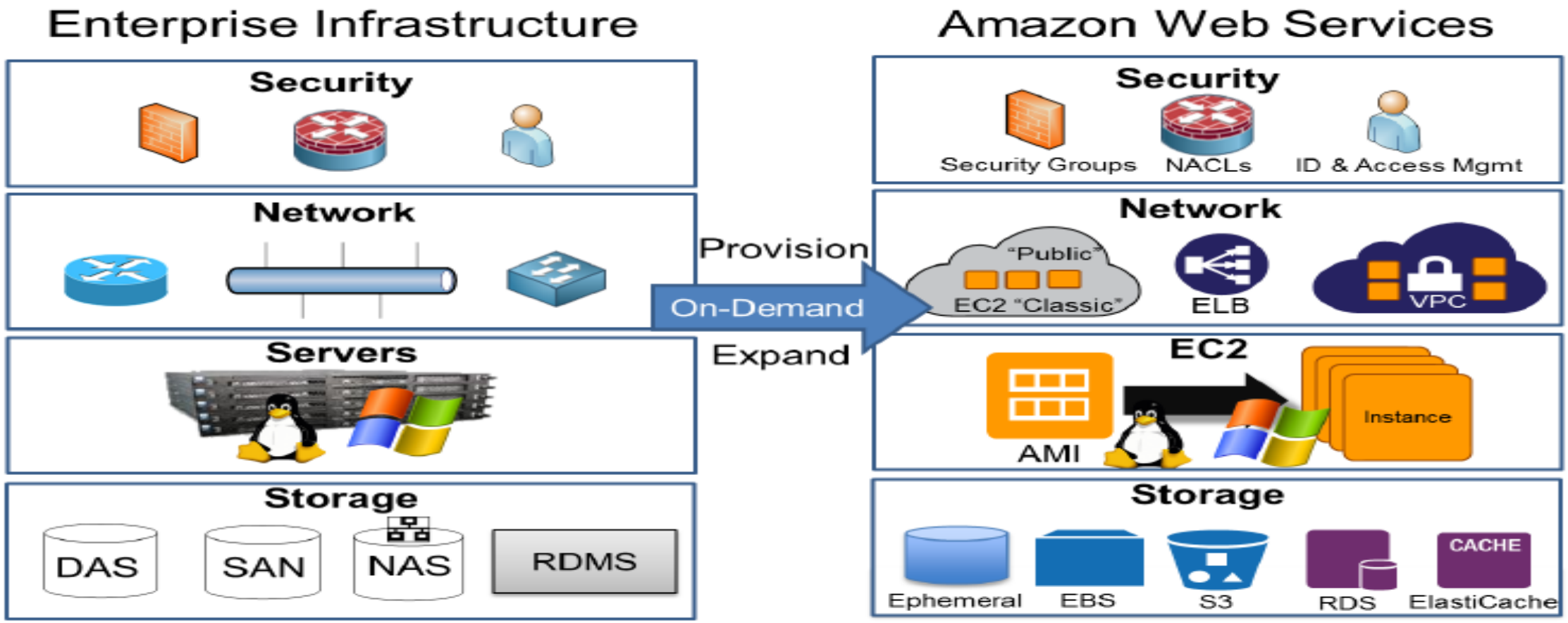
4 Regions

13 AZs

Amazon Web Services



AWS


Enterprise Infrastructure
Security
Network
Servers
Storage
DAS
SAN
NAS
RDMS
Provision
On-Demand
Expand
Amazon Web Services
Security
Security Groups
NACLs
ID & Access Mgmt
Network
"Public"
EC2 "Classic"
ELB
VPC
EC2
AMI
Instance
Storage
Ephemeral
EBS
S3
RDS
CACHE
ElastiCache

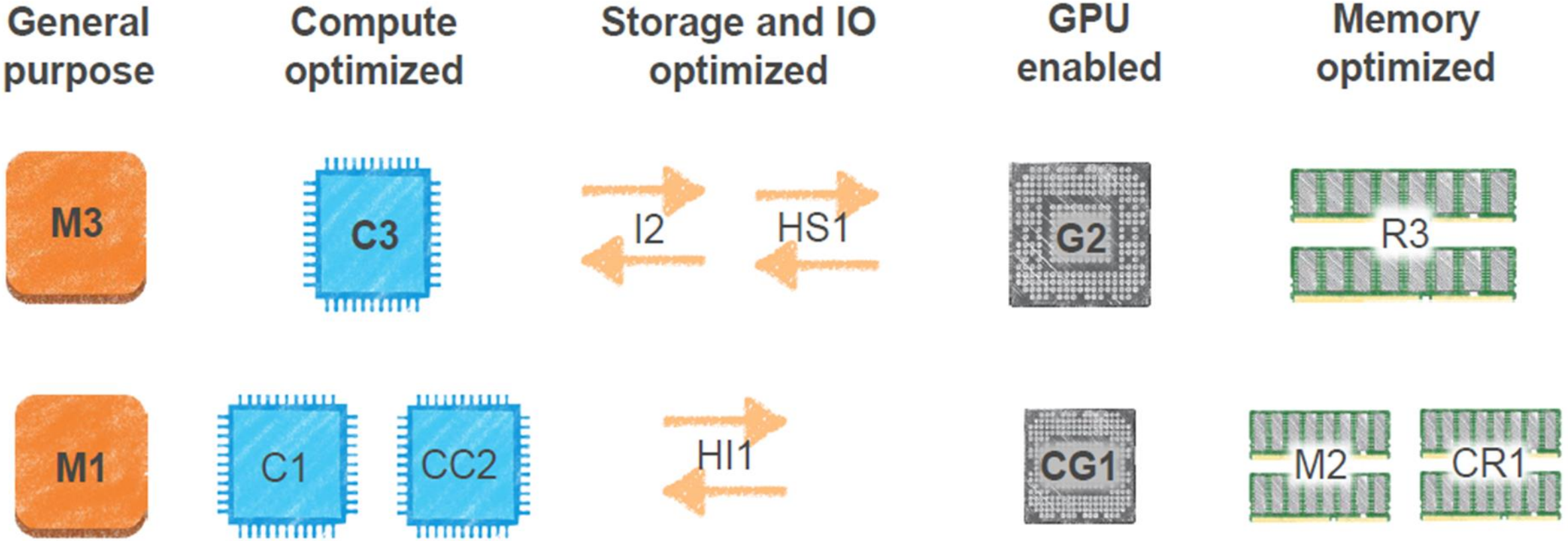
AWS
Compute Services

AWS Elastic Compute Cloud

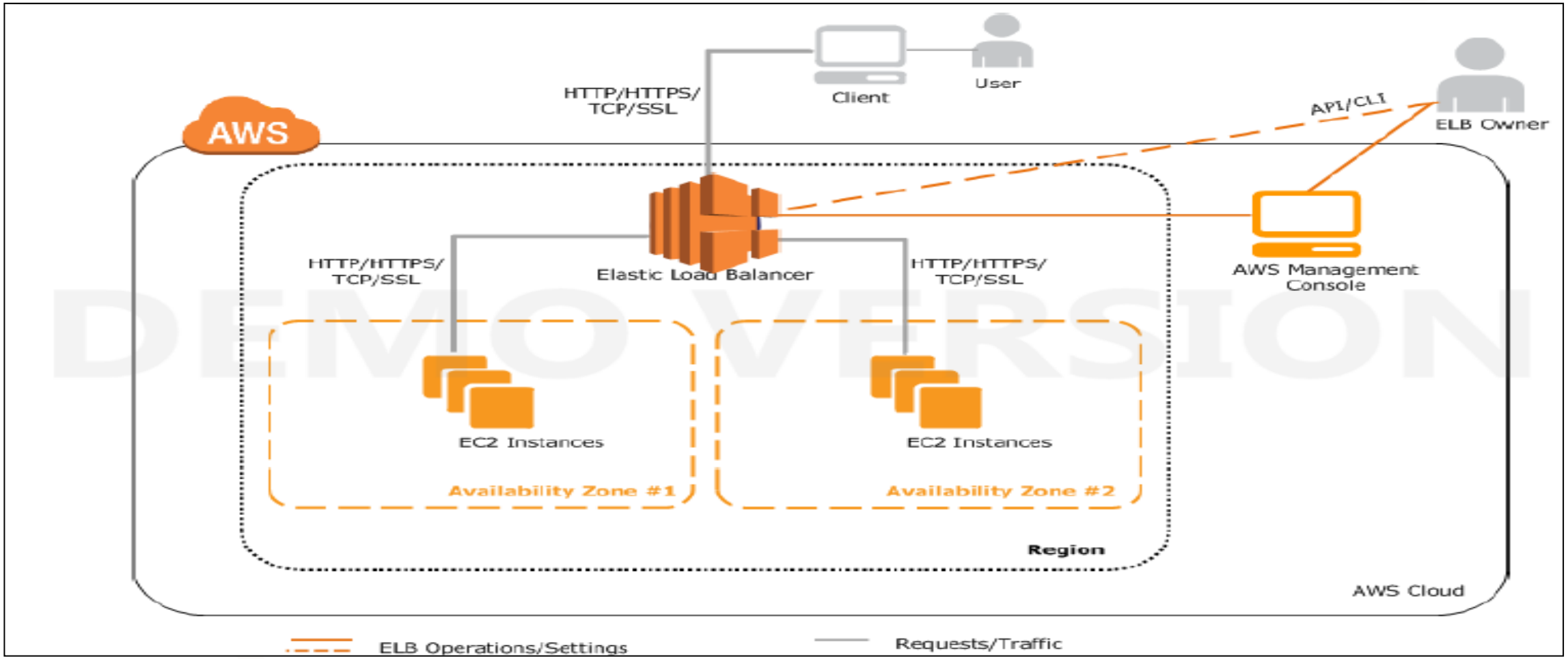
- Amazon EC2 stands for Elastic Compute Cloud, and is the Primary AWS web service.
- Provides Resizable compute capacity
- Reduces the time required to obtain and boot new server instances to minutes
- There are two key concepts to Launch instances in AWS:
  - Instance Type
  - AMI

- EC2 Facts:
  - Scale capacity as your computing requirements change
  - Pay only for capacity that you actually use
  - Choose Linux or Windows OS as per need. You have to Manage the OS and Security of same.
  - Deploy across AWS Regions and Availability Zones for reliability/HA

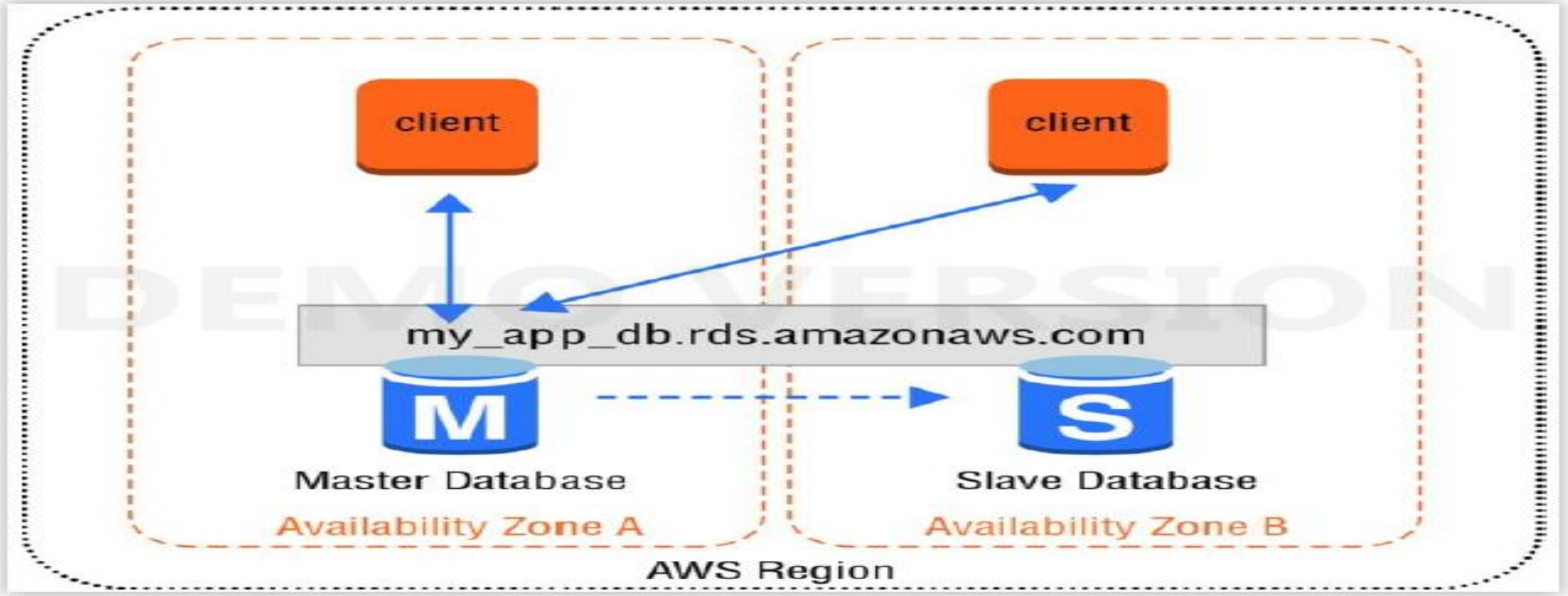
AWS EC2


General purpose
Compute optimized
Storage and IO optimized
GPU enabled
Memory optimized
M3
C3
I2
HS1
G2
R3
M1
C1
CC2
HI1
CG1
M2
CR1

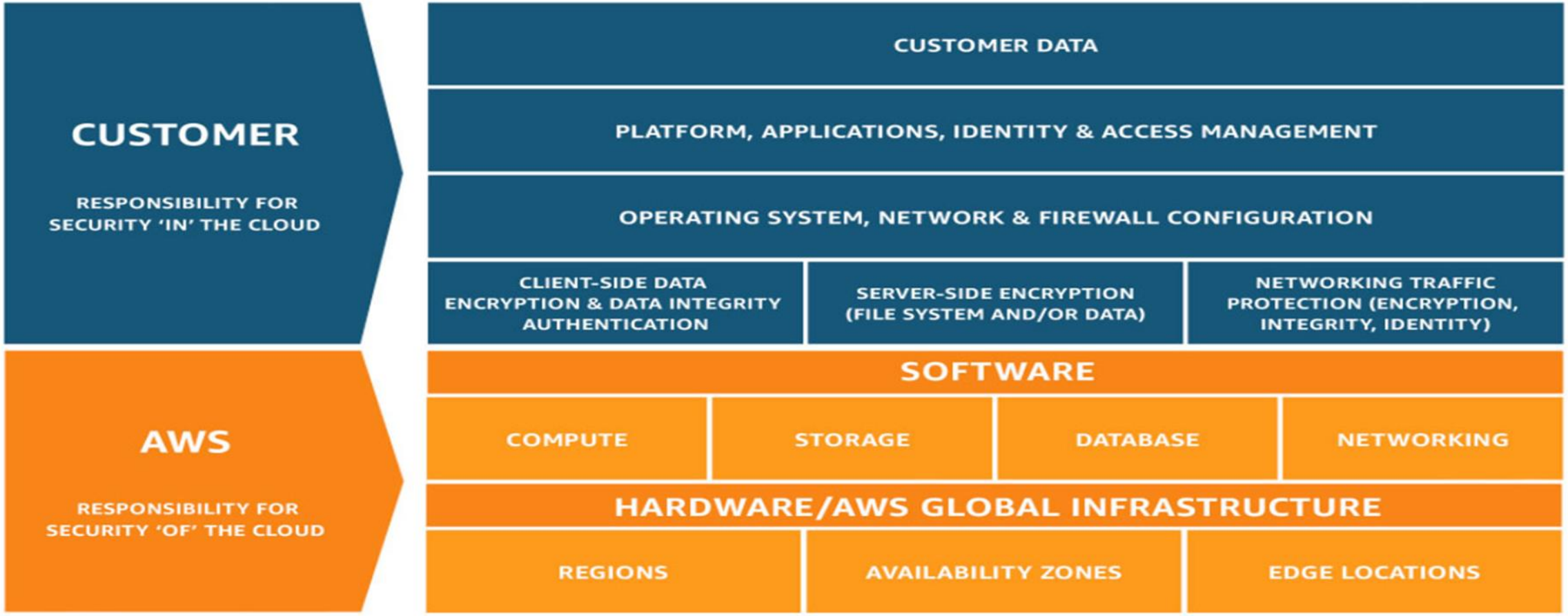
ELB



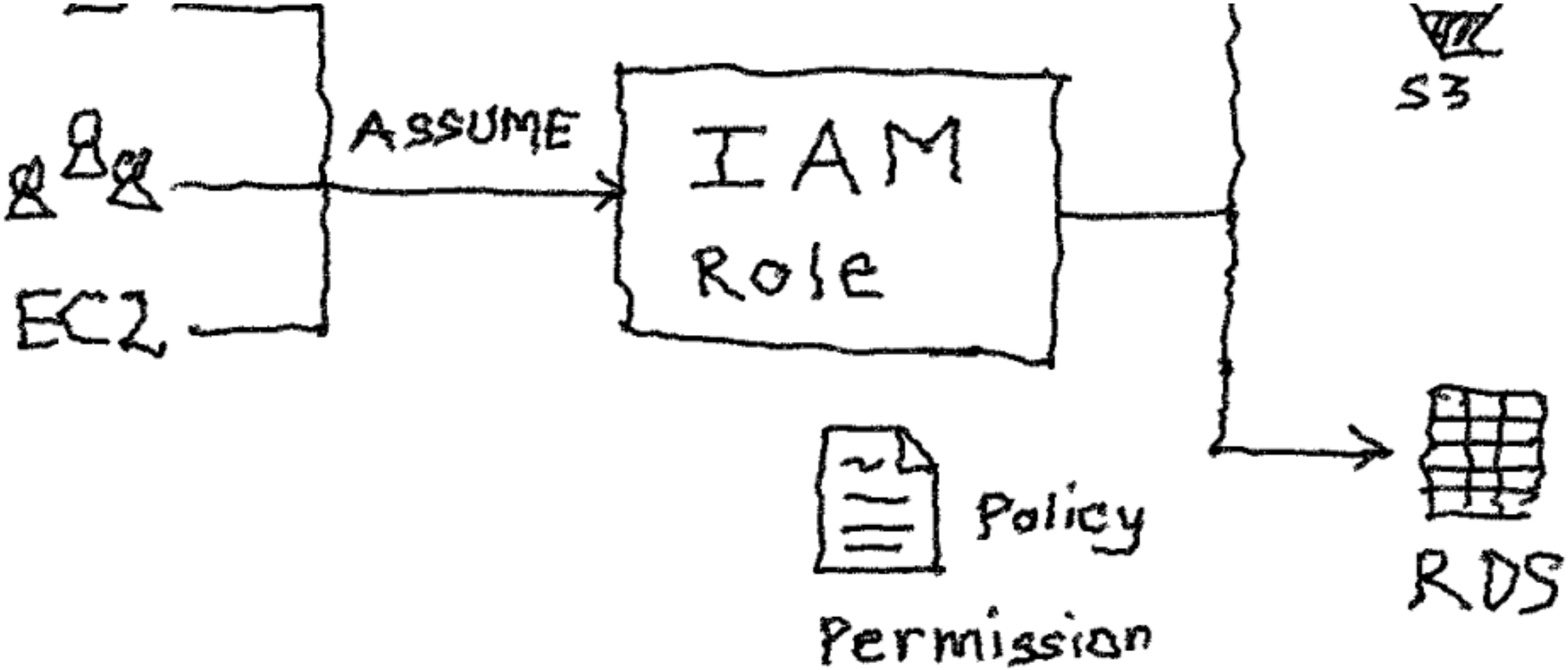
PaaS Example



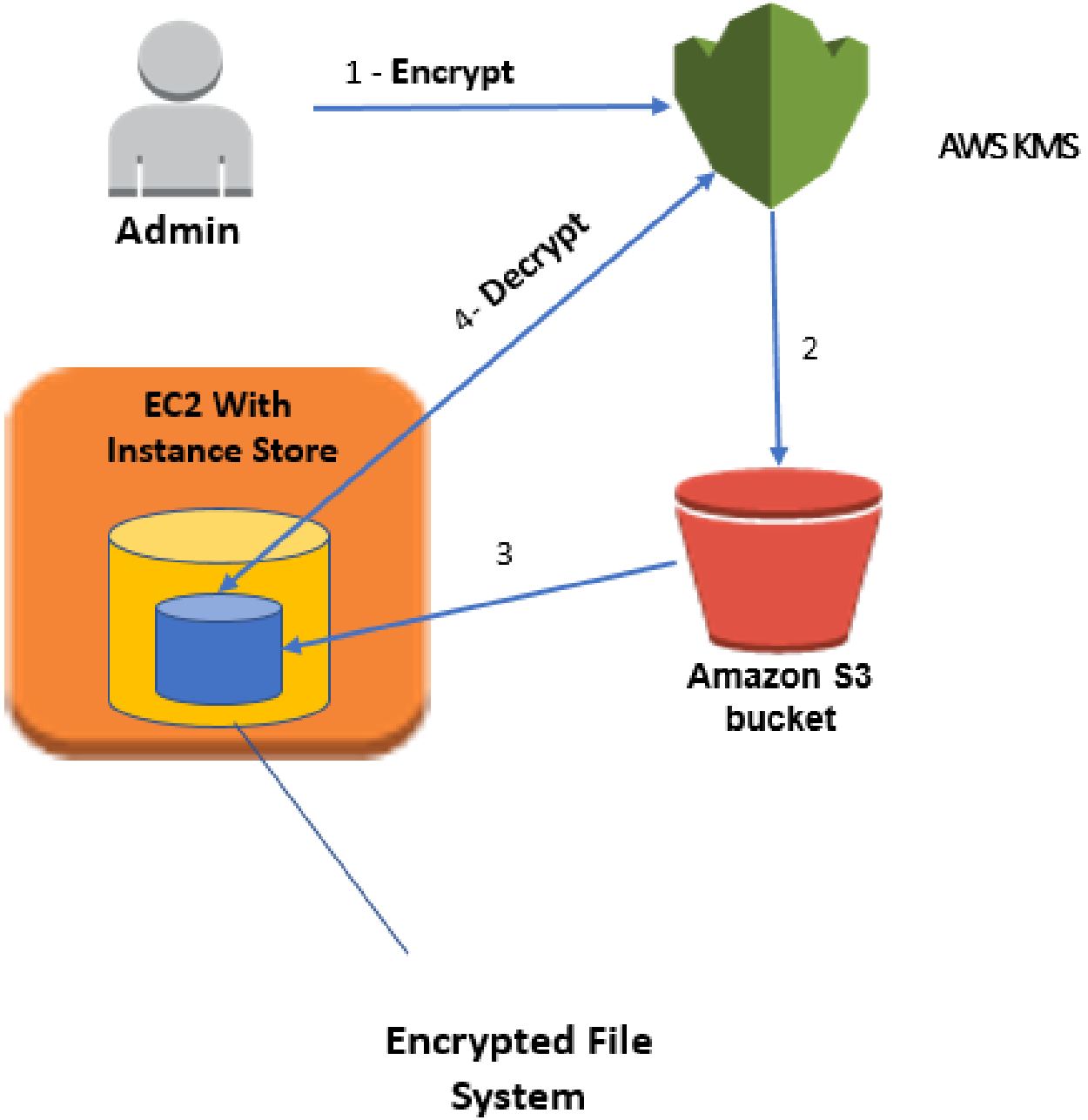
Shared Responsibility



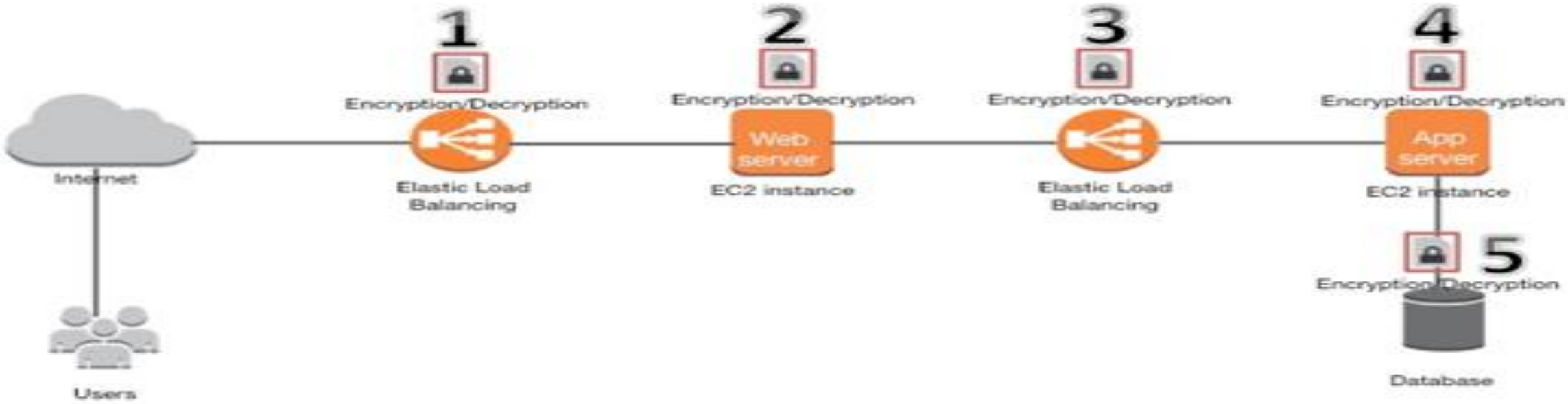
> *LAB : Creation of an ubuntu server and demonstation of security at instance level.*


ASSUME
IAM
Role
EC2
S3
RDS
Policy
Permission

LAB : Handling security credentials and IAM roles for security purpose.


1 - Encrypt
AWS KMS
Admin
4- Decrypt
2
EC2 With
Instance Store
3
Amazon S3
bucket
Encrypted File
System

End-to-end encryption in a standard web application



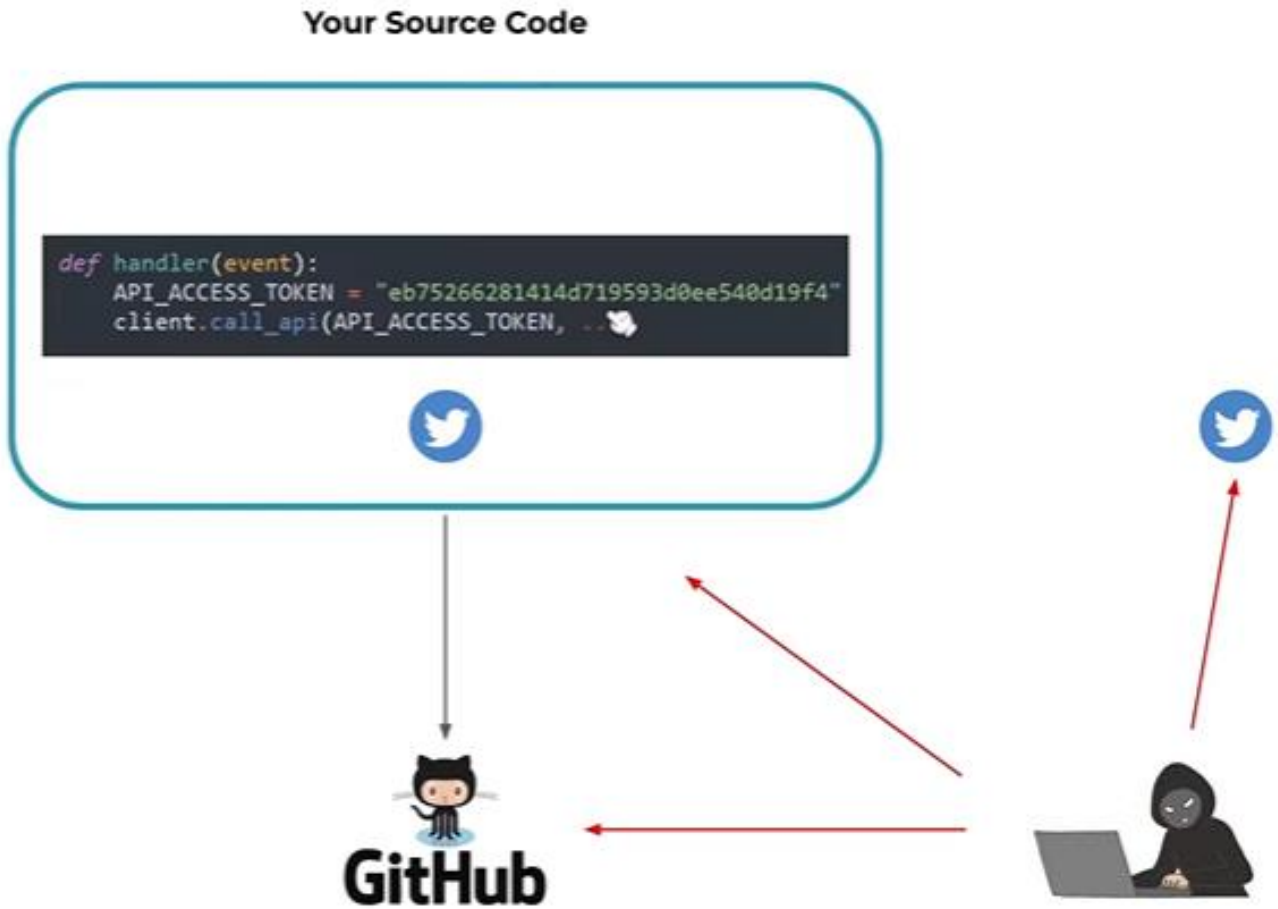
1. **Security Groups and NACLs**: Demonstrate how to use security groups and network access control lists (NACLs) to control inbound and outbound traffic to EC2 instances. Implement security best practices such as least privilege access.
2. **Identity and Access Management (IAM)**: Set up IAM roles and policies to control access to EC2 instances and other AWS resources. Use IAM roles for EC2 instances to grant permissions to interact with AWS services securely.
3. **Encryption**: Implement encryption for data at rest and data in transit. Use AWS Key Management Service (KMS) to manage encryption keys and encrypt EBS volumes or S3 buckets used by EC2 instances.
4. **Monitoring and Logging**: Configure CloudWatch alarms to monitor EC2 instance metrics such as CPU utilization, network traffic, and disk usage. Set up CloudWatch Logs to centralize logs from EC2 instances for troubleshooting and analysis.

4

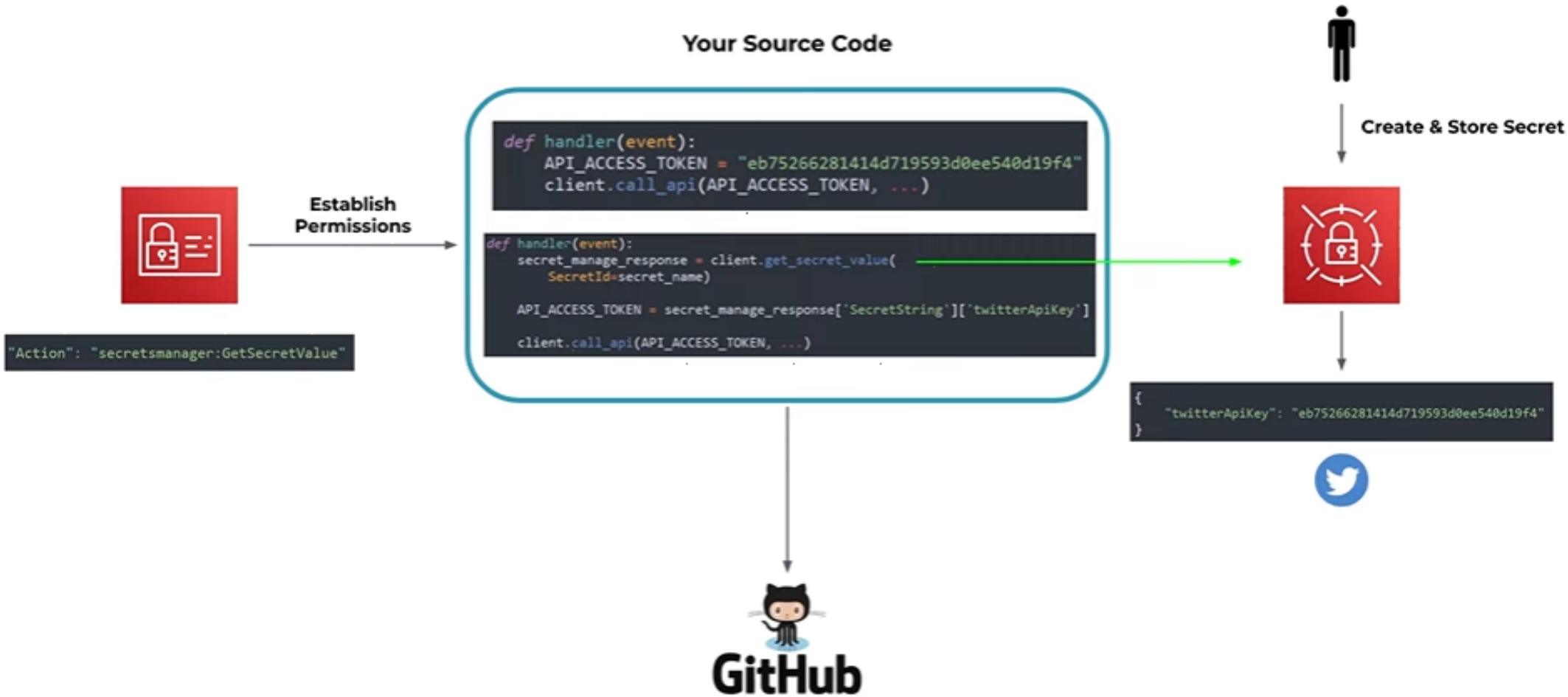
What is Secrets Manager?

- Managed AWS Service that **protects** api keys, tokens, database creds, ...
- Centralizes secret storage and **encrypts** your data
- Access is managed through IAM and monitored via Cloudwatch
- Allows you to control the **lifecycle** of secrets including **rotation**

An Example


Your Source Code
def handler(event):
API_ACCESS_TOKEN = "eb75266281414d719593d0ee540d19f4"
client.call_api(API_ACCESS_TOKEN, ...
GitHub

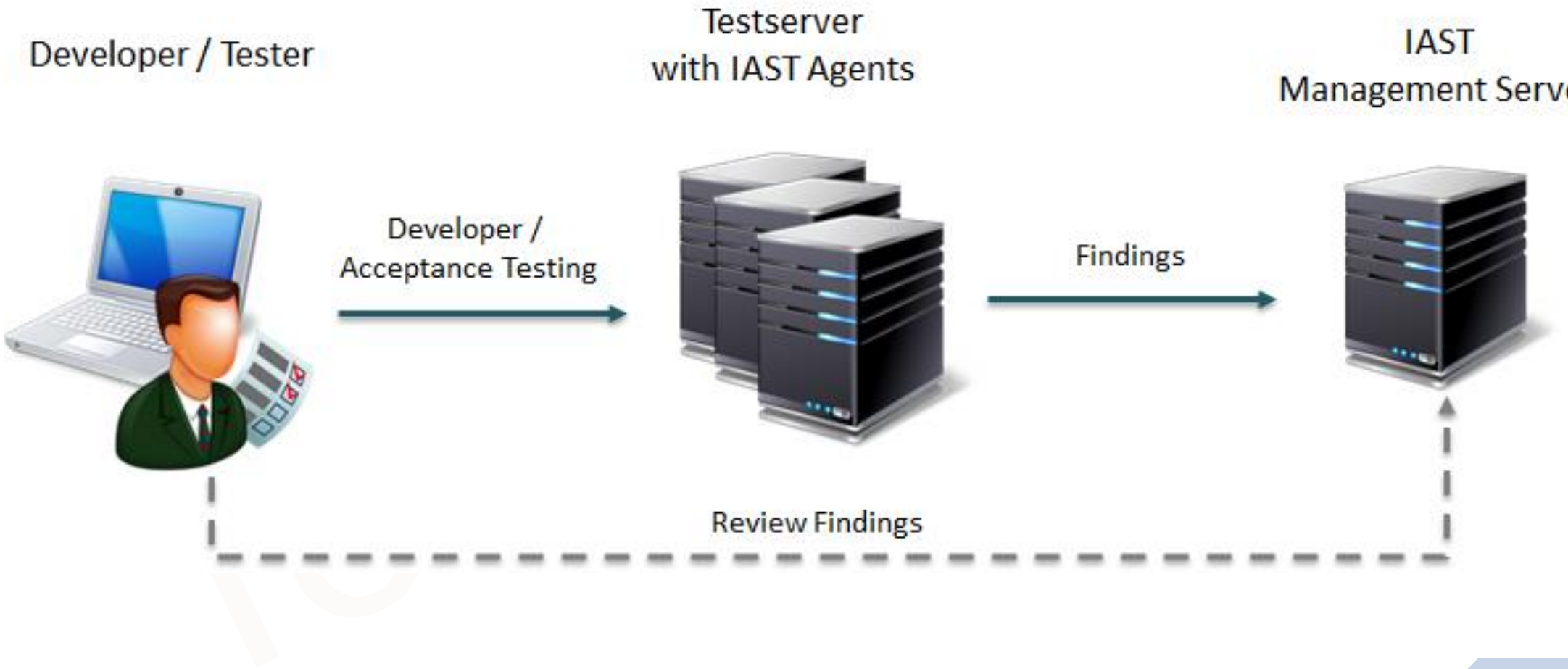
How Secrets Manager Helps



SAST vs. DAST

Static application security testing (SAST) and dynamic application security testing (DAST) are both methods of testing for security vulnerabilities, but they're used very differently.

SAST	DAST
White box security testing The tester has access to the underlying framework, design, and implementation. The application is tested from the inside out. This type of testing represents the developer approach.	**Black box security testing**The tester has no knowledge of the technologies or frameworks that the application is built on. The application is tested from the outside in. This type of testing represents the hacker approach.
Requires source codeSAST doesn't require a deployed application. It analyzes the sources code or binary without executing the application.	**Requires a running application**DAST doesn't require source code or binaries. It analyzes by executing the application.
Finds vulnerabilities earlier in the SDLC The scan can be executed as soon as code is deemed feature-complete.	**Finds vulnerabilities toward the end of the SDLC**Vulnerabilities can be discovered after the development cycle is complete.
Less expensive to fix vulnerabilities Since vulnerabilities are found earlier in the SDLC, it's easier and faster to remediate them. Findings can often be fixed before the code enters the QA cycle.	**More expensive to fix vulnerabilities**Since vulnerabilities are found toward the end of the SDLC, remediation often gets pushed into the next cycle. Critical vulnerabilities may be fixed as an emergency release.
Can't discover run-time and environment-related issues Since the tool scans static code, it can't discover run-time vulnerabilities.	**Can discover run-time and environment-related issues**Since the tool uses dynamic analysis on an application, it is able to find run-time vulnerabilities.
Typically supports all kinds of software Examples include web applications, web services, and thick clients.	**Typically scans only apps like web applications and web services**DAST is not useful for other types of software.


Developer / Tester
Testserver
with IAST Agents
IAST
Management Server
Developer /
Acceptance Testing
Findings
Review Findings

SAST	DAST	IAST
White box testing	Black box testing	Grey box testing
Does Source Code Analysis	Finds runtime vulnerability	Does both
Can be integrated into SDLC	Not integrated into SDLC	Can be integrated into SDLC
Used in the early stage of SDLC	Used at the later stage of SDLC	Used at both the stages of SDLC
Less expensive to fix the found vulnerability	Expensive to fix the found vulnerability	Depends on the vulnerability
Has false positives	Low false positives	Low false positives

Monitoring Fundamentals

- Monitoring refers to the process of observing and tracking various parameters, activities, or systems to ensure they are performing as expected. In various contexts, monitoring plays a crucial role in maintaining efficiency, security, and reliability.

1. **Early Detection of Issues**
2. **Optimization and Performance Improvement**
3. **Resource Utilization**
4. **Security and Compliance**
5. **Capacity Planning**
6. **Enhanced Reliability and Availability**

Site Reliability Engineering

Way to SRE

Site Reliability Engineering (SRE) is a term (and associated job role) coined by Ben Treynor Sloss, a VP of engineering at Google.

SRE is a job role, a set of practices which are known to work at ground, and some beliefs that animate those practices.

SRE is coined around Reliability of the system.

In general, an SRE has particular expertise around the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of the service(s) they are looking after.

SRE implements interface DevOps.

SRE is hiring software engineers to run products and to create systems to accomplish the work that would otherwise be performed, often manually, by sysadmins.

Common to all SREs is the belief in and aptitude for developing software systems to solve complex problems.

Site Reliability Engineering

Way to SRE

SRE is a team of people who (a) will quickly become bored by performing tasks by hand, and (b) have the skill set necessary to write software to replace their previously manual work, even when the solution is complicated.

SRE is fundamentally doing work that has historically been done by an operations team, but using engineers with software expertise, and banking on the fact that these engineers are inherently both predisposed to, and have the ability to, design and implement automation with software to replace human labor.

By design, it is crucial that SRE teams are focused on engineering. Without constant engineering, operations load increases and teams will need more people just to keep pace with the workload

Google places a 50% cap on the aggregate "ops" work for all SREs—tickets, on-call, manual tasks, etc. This cap ensures that the SRE team has enough time in their schedule to make the service stable and operable via engineering tasks of automation.

In general, an SRE team is responsible for the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of their service(s).

Important terms

Way to SRE

Availability=The ability of less downtime, or the fraction of the time that a service is usable. Although 100% availability is impossible, near-100% availability is often readily achievable

Reliability=The ability to work properly (even if some parts/components failed).

Durability=The ability of not losing data. Or the likelihood that data will be retained over a long period of time—is equally important (alike Availability) for data storage systems.

SLA (Promise) = Service-Level Agreement is a commitment between a service provider and a client, regarding particular aspects of the service – quality, availability, responsibilities etc. It is an explicit or implicit contract with your users that includes consequences of meeting (or missing) the SLOs they contain.

SLO (Goal) = Service Level Objective – The Objectives (within SLA, i.e. uptime, response time) which your team must hit to meet the SLA.

SLI (How and What) = Service Level Indicators – the Real numbers to measure your compliance against SLO. In Specific – its a carefully defined quantitative measure of some aspect of the level of service that is provided. i.e. request latency, error rate etc.

Important terms

Way to SRE

Availability %	Downtime per year[note 1]	Downtime per month	Downtime per week	Downtime per day
90% ("one nine")	**36.53 days**	**73.05 hours**	**16.80 hours**	**2.40 hours**
95% ("one and a half nines")	18.26 days	36.53 hours	8.40 hours	1.20 hours
97%	10.96 days	21.92 hours	5.04 hours	43.20 minutes
98%	7.31 days	14.61 hours	3.36 hours	28.80 minutes
99% ("two nines")	**3.65 days**	**7.31 hours**	**1.68 hours**	**14.40 minutes**
99.5% ("two and a half nines")	1.83 days	3.65 hours	50.40 minutes	7.20 minutes
99.8%	17.53 hours	87.66 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	**8.77 hours**	**43.83 minutes**	**10.08 minutes**	**1.44 minutes**
99.95% ("three and a half nines")	4.38 hours	21.92 minutes	5.04 minutes	43.20 seconds
99.99% ("four nines")	**52.60 minutes**	**4.38 minutes**	**1.01 minutes**	**8.64 seconds**
99.995% ("four and a half nines")	26.30 minutes	2.19 minutes	30.24 seconds	4.32 seconds
99.999% ("five nines")	**5.26 minutes**	**26.30 seconds**	**6.05 seconds**	**864.00 milliseconds**
99.9999% ("six nines")	**31.56 seconds**	**2.63 seconds**	**604.80 milliseconds**	**86.40 milliseconds**
99.99999% ("seven nines")	**3.16 seconds**	**262.98 milliseconds**	**60.48 milliseconds**	**8.64 milliseconds**
99.999999% ("eight nines")	**315.58 milliseconds**	**26.30 milliseconds**	**6.05 milliseconds**	**864.00 microseconds**
99.9999999% ("nine nines")	**31.56 milliseconds**	**2.63 milliseconds**	**604.80 microseconds**	**86.40 microseconds**

Monitoring

Way to SRE

Observability
Although not strictly defined, observability is a general term used to describe processes and techniques related to increasing awareness and visibility into systems. This can include monitoring, metrics, visualization, tracing, and log analysis.

Monitoring
Collecting, processing, aggregating, and displaying real-time quantitative data about a system, such as query counts and types, error counts and types, processing times, and server lifetimes.

Alerting
Alerting is the responsive component of a monitoring system that performs actions based on changes in metric values. A notification intended to be read by a human and that is pushed to a system such as a bug or ticket queue, an email alias, or a pager. Respectively, these alerts are classified as tickets, email alerts, and pages.

Logging
Logging is the practice of managing all of the log data produced by your applications and infrastructure. Logging process consists of many processes like Log aggregation, Log shipping, Storage and Archiving, Log Enrichment, Log Security, Log Analysis etc.

Visualization
Visualization is the process of presenting metrics data in a format that allows for quick, intuitive interpretation through graphs or charts.

Monitoring

Way to SRE

White-box monitoring
Monitoring based on metrics exposed by the internals of the system, including logs, interfaces like the Java Virtual Machine Profiling Interface, or an HTTP handler that emits internal statistics.

Black-box monitoring
Testing externally visible behavior as a user would see it.

Latency
Latency is a measure of the time it takes to complete an action. Depending on the component, this can be a measure of processing, response, or travel time.

Throughput
Throughput represents the maximum rate of processing or traversal that a system can handle.

Performance
Performance is a general measure of how efficiently a system is completing work. Performance is an umbrella term that often encompasses work factors like throughput, latency, or resource consumption.

Saturation
Saturation is a measure of the amount of capacity being used. Full saturation indicates that 100% of the capacity is currently in use.

Monitoring

Way to SRE

Data point
A data point is a single measurement of a single metric

Data set
A data set is a collection of data points for a metric.

Time series
Time series data is a series of data points that represent changes over time.

Sampling rate
Sample rate is a measurement of how often a representative data point is collected in lieu of continuous collection. A higher sampling rate more accurately represents the measured behavior but requires more resources to handle the extra data points.

Threshold
When alerting, a threshold is the boundary between acceptable and unacceptable values which triggers an alert if exceeded.

Trend
A trend is the general direction that a set of values is indicating. Trends are more reliable than single values in determining the general state of the component being tracked.

Why Monitor?

Way to SRE

Analyzing long-term trends

Comparing over time

Alerting

Building dashboards

Conducting ad hoc retrospective analysis

Service Level Objectives

Way to SRE

It's impossible to manage a service correctly, let alone well, without understanding which behaviors really matter for that service and how to measure and evaluate those behaviors.

We use intuition, experience, and an understanding of what users want to define service level indicators (SLIs), objectives (SLOs), and agreements (SLAs). These measurements describe basic properties of metrics that matter, what values we want those metrics to have, and how we'll react if we can't provide the expected service. Ultimately, choosing appropriate metrics helps to drive the right action if something goes wrong, and also gives an SRE team confidence that a service is healthy.

An SLO is a service level objective: a target value or range of values for a service level that is measured by an SLI. A natural structure for SLOs is thus SLI ≤ target, or lower bound ≤ SLI ≤ upper bound.

Choosing an appropriate SLO is usually complex (i.e. QPS, Network Bandwidth etc.) , but sometime its straightforward too (i.e. setting low-latency).

Choosing and publishing SLOs to users sets expectations about how a service will perform. Without an explicit SLO, users often develop their own beliefs about desired performance, which may be unrelated to the beliefs held by the people designing and operating the service.

Monitoring with proper SLI Metric

Way to SRE

Always make sure to select the right Service Level Indicator Metric to track and alert.

Set alerts (with respective criticality, pager, Email etc) for all your SLO targets and also creating simple and meaning full dashboards for a higher visibility.

Four golden signals to track:

- Latency
- Traffic
- Errors
- Saturation (IO, Memory, CPU etc)

SLI – Indicators in Practice

Way to SRE

SRE doesn't typically get involved in constructing SLAs, however, get involved in helping to avoid triggering the consequences of missed SLOs.

What Do You and Your Users Care About?

- User-facing serving systems, such as the Shakespeare search frontends, generally care about availability, latency, and throughput. In other words: Could we respond to the request? How long did it take to respond? How many requests could be handled?
- Storage systems often emphasize latency, throughputs, IOPS, availability, and durability.
- Big data systems, such as data processing pipelines, tend to care about throughput and end-to-end latency (How much data is being processed?, and how long it takes to process?).
- All systems should care about correctness: was the right answer returned, the right data retrieved, the right analysis done?

Collecting Indicators

Aggregate the metric for better usage (Average out, Instantaneous usages)
Standardize Indicators (Over a period of time i.e. average packets per minute)
Collect Indicators at server as well as Client end

Objectives in Practice (SLO)

Define Objectives (99% (averaged over 1 minute) of Get RPC calls will complete in less than 100 ms)
Choose realistic targets, which are simple & minimum-required and always keep a refine strategy/scope.

Control Measures

Way to SRE

SLIs and SLOs are crucial elements in the control loops used to manage systems:

- Monitor and measure the system's SLIs.
- Compare the SLIs to the SLOs, and decide whether or not action is needed.
- If action is needed, figure out what needs to happen in order to meet the target.
- Take that action.

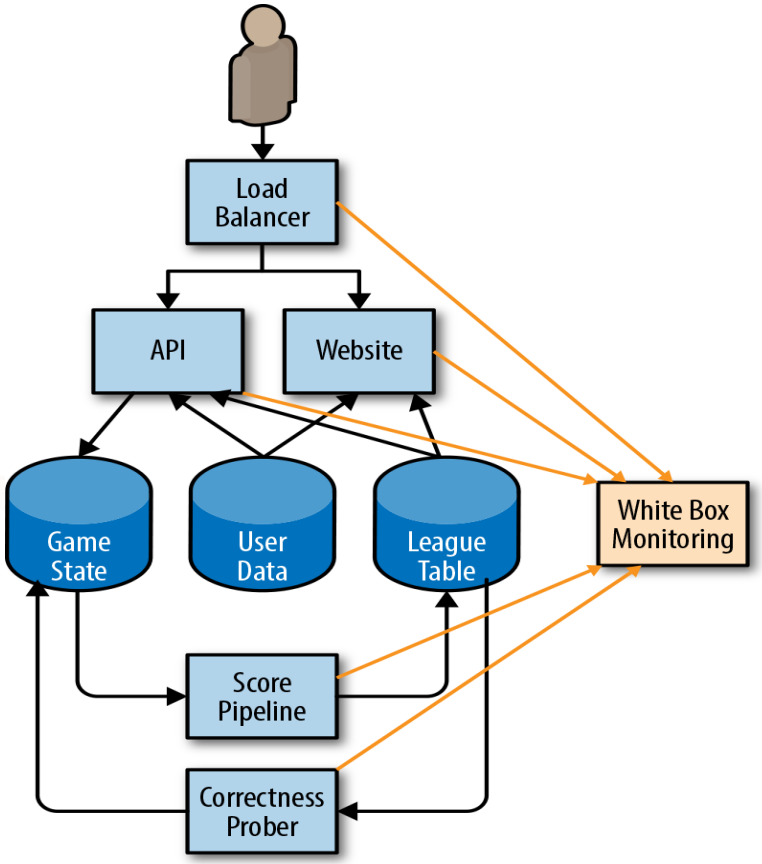
Always remember:

- Publishing SLOs sets expectations for system behavior
- Keep margins - Using a tighter internal SLO than the SLO advertised to users gives you room to respond to chronic problems before they become visible externally.
- If your service's actual performance is much better than its stated SLO, users will come to rely on its current performance. You can avoid over-dependence by deliberately taking the system offline occasionally (Google's Chubby service introduced planned outages in response to being overly available).

Understanding how well a system is meeting its expectations helps decide whether to invest in making the system faster, more available, and more resilient. Alternatively, if the service is doing fine, perhaps staff time should be spent on other priorities, such as paying off technical debt, adding new features, or introducing other products.

Monitoring in place

Way to SRE



Error Budget and policies

Way to SRE

The SLO is a target percentage, and the error budget is 100% minus the SLO. For example, if you have a 99.9% success ratio SLO, then a service that receives 3 million requests over a four-week period had a budget of 3,000 (0.1%) errors over that period. If a single outage is responsible for 1,500 errors, that error costs 50% of the error budget.

Once you have an SLO, you can use the SLO to derive an error budget. In order to use this error budget, you need a policy outlining what to do when your service runs out of budget.

When we talk about enforcing an error budget policy, we mean that once you exhaust your error budget (or come close to exhausting it), you should do something in order to restore stability to your system

Common owners and actions might include:

- The development team gives top priority to bugs relating to reliability issues over the past four weeks.
- The development team focuses exclusively on reliability issues until the system is within SLO. This responsibility comes with high-level approval to push back on external feature requests and mandates.
- To reduce the risk of more outages, a production freeze halts certain changes to the system until there is sufficient error budget to resume changes.

Monitoring Aspects

IT infrastructure Monitoring is a wider area that covers below major tools/assets:

- Server (Physical & Virtual) monitoring (CPU, memory, Battery, Hypervisor, Hardware, Temp, Fan Speed etc)
- Operating System Monitoring – (CPU/Memory, Processes, Zombie, OS Config Changes, Logins etc)
- Network Monitoring (Availability, Bandwidth, Packets, Traffic, Latency, Packet Loss etc)
- Application Monitoring (Availability, Performance, Transactions, Failure messages, Timeouts, RCA, Latency etc)
- Storage Monitoring - SAN arrays, LUNs, storage pools/RAID groups, IOPS, Throughputs etc.
- Distributed tracing – It is helpful for monitoring the execution of an application at the level of the code itself, rather than the infrastructure that hosts it.
- Synthetic and Real-User Monitoring – Simulated and Real-user monitoring for customers' digital experiences by looking at how online visitors are interacting with a website or application, analyzing everything from page load events to AJAX requests to Apdex Score and frontend application crashes, uptime, certificate issues, load times, APIs, and so much more.

Infra Monitoring - Challenges

- License Management
- Hybrid Environment (Cloud)
- Container Monitoring

Tools Available in Market

- Nagios
- Prometheus
- SolarWinds Server & Application Monitor (SAM)
- Zabbix
- Datadog
- Dynatrace
- AWS Cloudwatch/Azure Monitor/ GCP Stack driver
- cAdvisor
- Grafana

Cloud In-House Monitoring Tools – Cloudwatch

CloudWatch

- A monitoring service for AWS cloud resources and the applications you run on AWS
- Visibility into resource utilization, operational performance, and overall demand patterns
- Custom application-specific metrics of your own
- Accessible via AWS Management Console, APIs, SDK, or CLI
- Monitor other AWS resources
- View graphics and statistics
- Container Insights
- Synthetics
- Log Groups and Insights
- Set Alarms

RDS Performance Insights

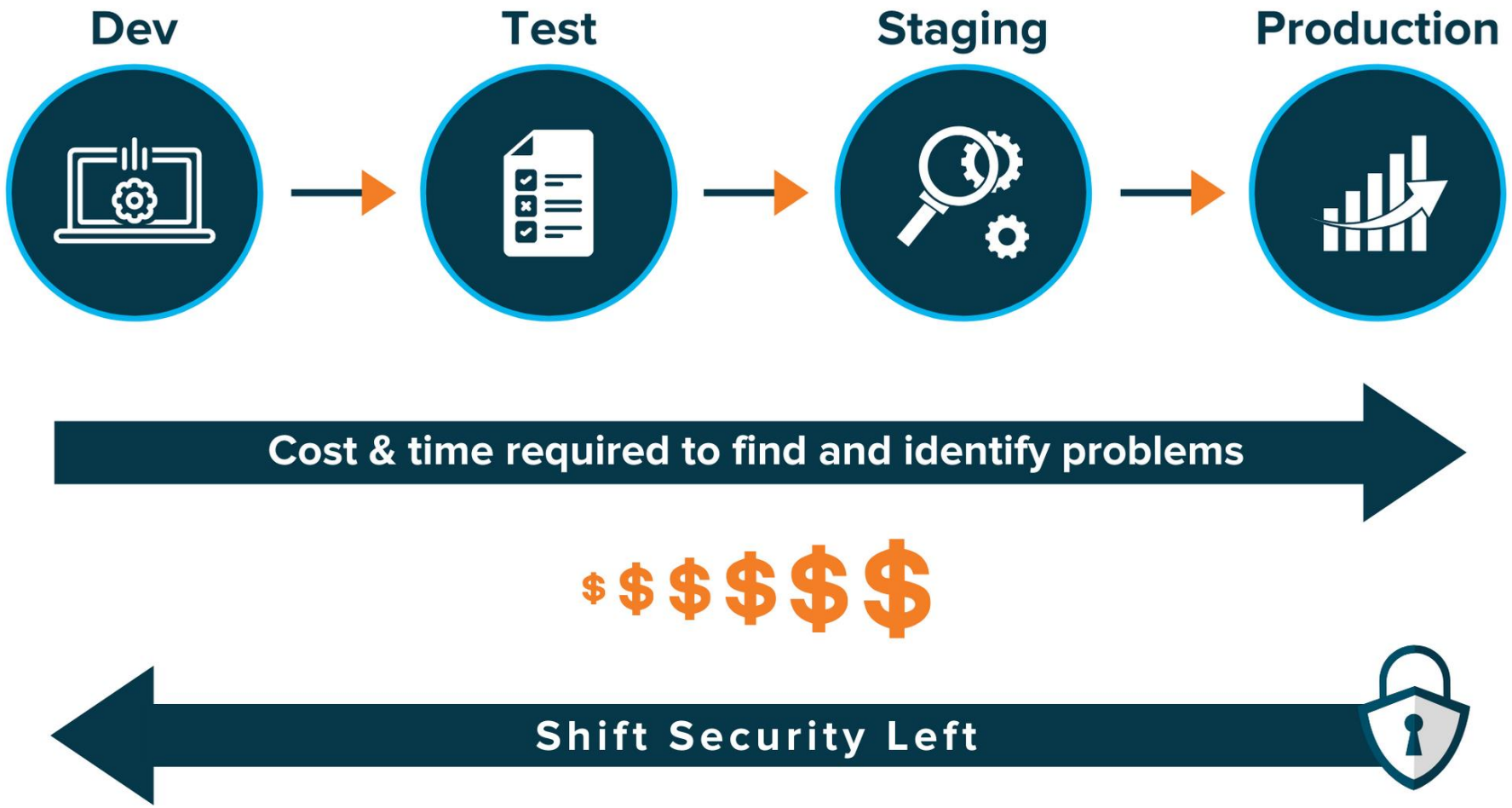
- Amazon RDS Performance Insights monitors your Amazon RDS DB instance load so that you can analyse and troubleshoot your database performance.
- It provides you a database performance dashboard where you can quickly assess performance on relational database workloads.
- Currently available for some specific versions of Amazon Aurora PostgreSQL compatible Edition, MySQL compatible Edition, PostgreSQL, MySQL, and Oracle.
- If you have more than one database on the DB instance, performance data for all of the databases is aggregated for the DB instance.
- You can gain insight of your database with “Performance Insight” on below area:
  - Top SQL queries causing load on DB
  - Average Active session
  - hosts
  - users

Cloud In-house Tools

- You can’t avoid CloudWatch/Cloud-inhouse monitoring tools in case of PaaS Services
- These tools are expensive if used for On-Prem Solutions
- Internet access is required for working with these tools

Logging vs Monitoring

sematext


Dev
Test
Staging
Production
Cost & time required to find and identify problems
$ $ $ $ $ $
Shift Security Left

CIA terms of DevSecOps:

1.Confidentiality:

1. **Data Encryption:** Implement encryption mechanisms to protect sensitive data both in transit and at rest. This ensures that only authorized individuals can access and view confidential information.
2. **Access Controls:** Employ robust access control mechanisms to restrict access to confidential resources based on roles and permissions. This prevents unauthorized users from accessing sensitive data.

2.Integrity:

1. **Code Integrity Checks:** Integrate automated tools for code signing and verification to ensure that code remains unchanged and uncorrupted throughout the development pipeline.
2. **Version Control:** Utilize version control systems such as Git to track changes made to code and ensure that only authorized changes are accepted. This helps maintain the integrity of the codebase.

3.Availability:

1. **High Availability Architectures:** Design resilient and fault-tolerant architectures to ensure continuous availability of applications and services. Implement redundancy, load balancing, and failover mechanisms to minimize downtime.
2. **Automated Monitoring:** Implement proactive monitoring and alerting systems to detect and mitigate potential issues that could affect system availability. This includes monitoring server health, network traffic, and application performance metrics.

- The OWASP (Open Web Application Security Project) Top 10 list outlines the most critical security risks for web applications. These vulnerabilities are commonly exploited by attackers to compromise the confidentiality, integrity, and availability of web applications.

1. **Injection:**
    1. Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query. This can lead to various types of attacks, including SQL injection, LDAP injection, and OS command injection.
2. **Broken Authentication:**
    1. Broken authentication vulnerabilities occur when authentication mechanisms are implemented incorrectly or are weak. This can lead to unauthorized access to sensitive data or functionality.
3. **Sensitive Data Exposure:**
    1. Sensitive data exposure vulnerabilities occur when sensitive information, such as passwords or credit card numbers, is not adequately protected. This can result from improper encryption, insecure storage, or inadequate access controls.
4. **XML External Entities (XXE):**
    1. XXE vulnerabilities occur when an application processes XML input containing external entity references. Attackers can exploit these vulnerabilities to access local or remote files, execute arbitrary code, or perform denial-of-service (DDOS) attacks.
5. **Broken Access Control:**
    1. Broken access control vulnerabilities occur when access controls are not enforced properly, allowing unauthorized users to access sensitive functionality or data. This can result from missing or ineffective access control checks.