NAYA College

Introduction to DevOps

Lev Epshtein

About me

Lev Epshtein

Technology enthusiast with more than 20 years of industry experience in DevOps and IT. Industry experience with back-end architecture.

Solutions architect with experience in big scale systems hosted on AWS/GCP, end-to-end DevOps automation process.

Cloud DevOps, and Big Data instructor.

Certified GCP trainer,

Partner & Solution Architect Consultant at Opsguru.

Co-Founder: https://www.firewave.earth/

lev@opsguru.io, lev@firewave.earth

NAYA College

- MAKING A CHANGE -

...

Module - DEVOPS TOOLBOX

- WHAT IS THE AGENDA BEHIND THIS SESSION -

For us to understand that:

- Devops is not a magic word
- Devops requires collaboration of the entire organization
- Going Devops is massive move but doing it right is usually the hardest part
- Familiarize with the design patterns that makes a devops what it is
- The company culture is usually the bottleneck in the transition to “the DevOps concept”
- Software architecture is a super critical part for achieving Devops originated organization

Agenda

- DEVOPS - INTRO
- Devops Hot Spots

- INTRO -
THE PROBLEM

- Everything is software BUT it still runs on Servers to operate…
- Delivering a service is painfully slow and full with Errors and faulty functionality
- Internal frictions between Product / R&D / IT creates the problem
- Delay in Delivery = Money
- The company culture is usually the bottleneck in the transition to “the DevOps concept”

NAYA College

SYMPTOMS

- Faulty software services are release into production - Outages are expected
- No KPI and Monitoring facilitators are in place to help diagnose production issues
- Quickly and effectively.
- No Automated regression and load Tests
- No proper incident management and alerting in place From Alert to Postmortem.
- Blaming and shifting fingers pointing
- Long iteration from request to response for resources required by groups such QA , DEV to other teams and vice versa.
- “Manual errors / Human error” will be in most cases the ROOT CAUSE.
- Features are being released as “enabled” to the wild
- Life as an Ops engineer sucks in your company.

- INTRO -
WALL OF CONFUSION

Developers and IT Operations has a very different objective

- INTRO -
THE OBJECTIVE

COMPANY OBJECTIVE

- IMPROVE TTM TIME TO MARKET -

TTM AGENDA

- Shorten the time from product to production delivery
- Cut down the time required for Technical and technological evolutions and implementations
- While keeping very high level of stability, quality, performance, cost and basically 0 downtime...

COMPANY OBJECTIVE

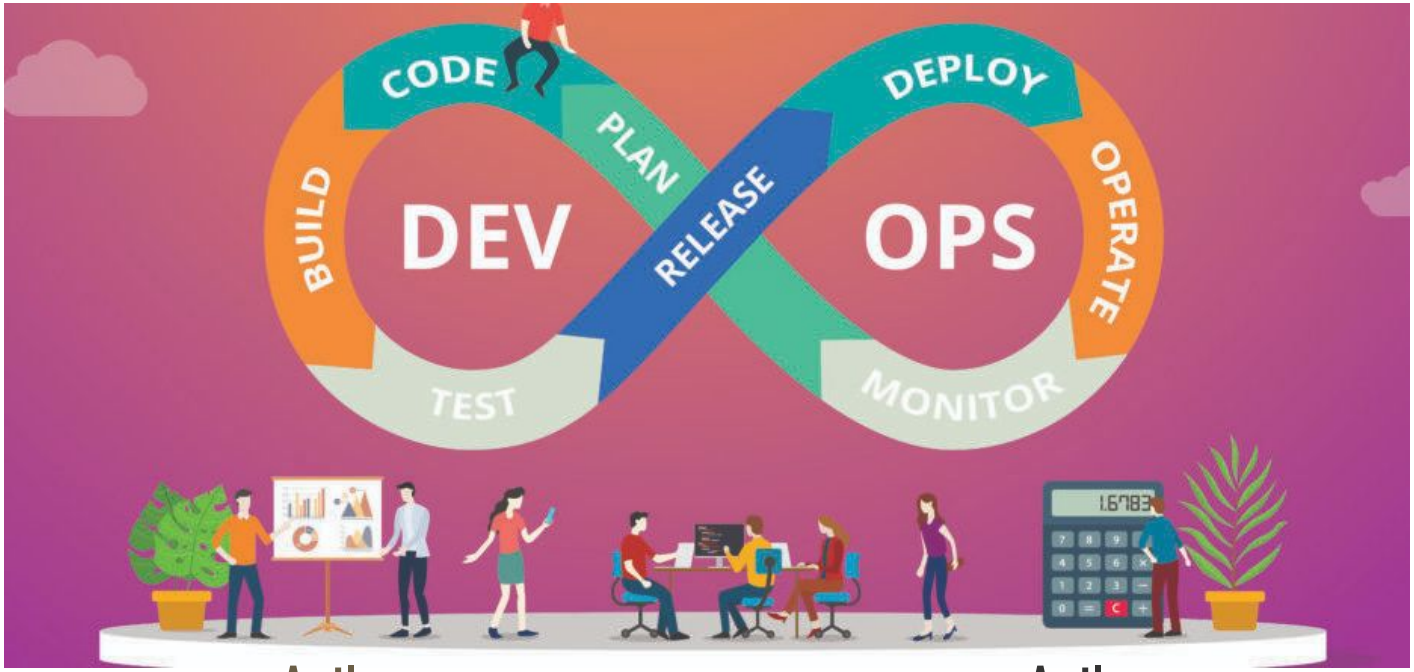
- IMPROVE TTM TIME TO MARKET -

TTM PROBLEM

Two Worlds collided

- Improving TTM and improve quality are two opposing attributes in the development arena.
- Stability improvement is the **objective of IT Ops teams**
- Reducing/Improving TTM is the **objective of developers**

SO HOW DO WE SOLVE IT?



Agile Development

Agile Deployment/Production

- INTRO -
SO WHAT IS DEVOPS?

DevOps was conceived from the idea of extending the Agile development practice.

Meaning:

Extending the software streamlining movement of Build , Validate and Deploy stages and tie it up with cross-functional teams from Design, Operation , Quality and Build automation, Delivery , Monitoring and thru production support.

In other words: Devops is not just one person or a methodologie,
A Devops is a culture and a set of mind where one is expected to think out of the box.
To plan , Build and maintain a infinie scalable, elastic and cloud native product (even if it's on-premise),
Software first and an IT second.

WHAT ARE THE EXPECTATION?

DevOps Culture

What do we expect to gain from a **DevOps** culture in our company and why...

Ideally

- Standardizing development environments
- IAC - Infrastructure as a code
- CD - Automate delivery process to improve delivery predictability , efficiency , security and maintainability
- Monitoring and Log shipping frameworks
- Culture of collaboration

NAYA College

Provide **Developers** with more control of the production environment and a better understanding of the production infrastructure

- MOVING FORWARD - DEVOPS HOTSPOTS

DevOps Foundation®

BLUEPRINT

With DevOps, people across the IT organization, working together, enable fast flow, feedback and continuous improvement of planned work into production, while achieving quality, stability, reliability, availability, security and team satisfaction.

CALMS Values

C: Culture - emphasizes shared vision collaboration, communication, learning and continuous improvement.
A: Automation - CI/CD toolchains, and infrastructure-as-code enable automation, consistency, velocity and fast recovery.
L: Lean - Maximize customer value while minimizing waste and improving flow.
M: Measurement - Value-driven metrics for people, process and technology support trust and performance improvement.
S: Sharing - Leaders and teams share ideas, and skills, improve communication, collaboration and performance.

The Three Ways

1st Way: Continuous Flow
2nd Way: Feedback
3rd Way: Continuous Improvement
(Experimenting and Learning)

Principles & Practices

Frequent small releases using continuous integration, testing, delivery, deployments and monitoring reduce lead time, costs and risks.

Organization

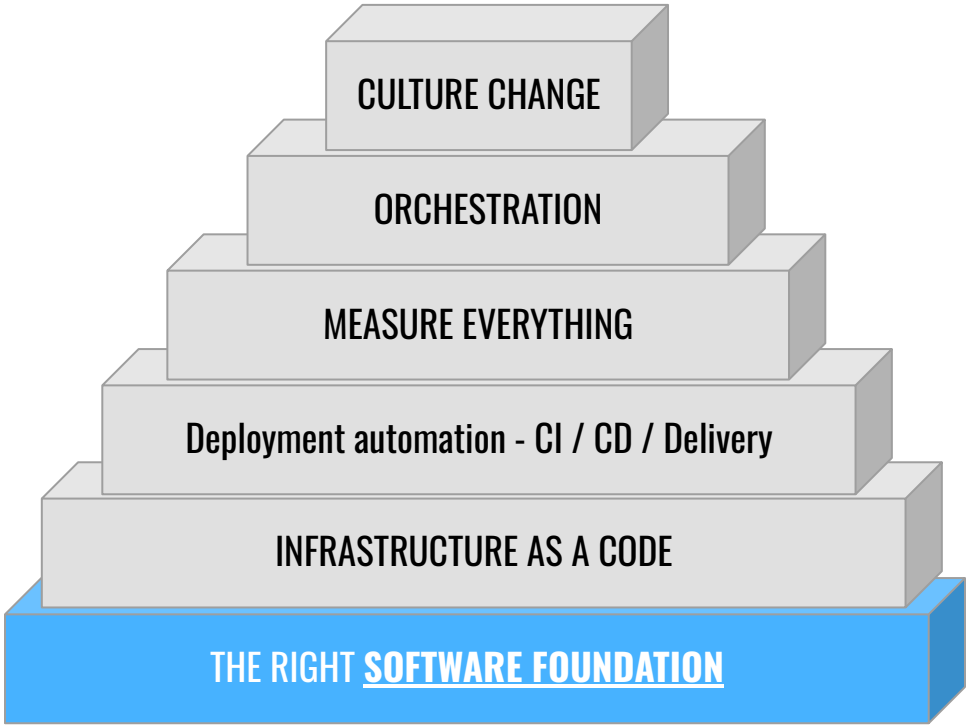
Cross-function teams focus on business goals. Ops work with Dev, supporting each other to improve flow towards production and monitor results.

Benefits

Improved release cadence, velocity, throughput, efficiency and stability, quality, security and team satisfaction.

Related Frameworks

Agile: Lean Development
ITSM: Processes
Lean: Reduce Waste
Value Stream Management: End-to-End


CULTURE CHANGE
ORCHESTRATION
MEASURE EVERYTHING
Deployment automation - CI / CD / Delivery
INFRASTRUCTURE AS A CODE
THE RIGHT SOFTWARE FOUNDATION

THE RIGHT SOFTWARE FOUNDATIONS

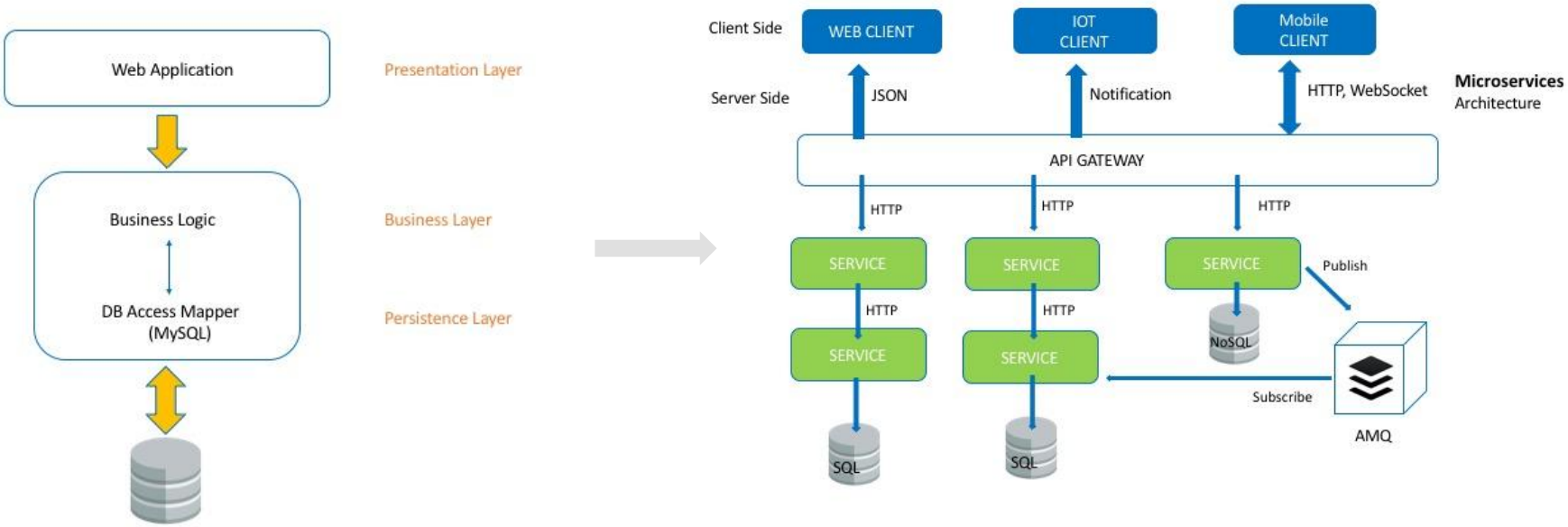
- Develop inside containers
  - Start early as possible

- Monolith ---> Microservices make the change by rebuilding not refactoring

- Build your software in the right design patterns
  - Microservices are not just splitting your code and using API
  - Go Async!
  - Implement Message queueing
  - Use your API as it was intended
  - Use discovery systems
  - Prepare for partial service outage handling
  - Use Small DB's per role / service
  - Go Async!

What is Microservices Architecture

microservice architecture is the “latest” (since 2014…) method of developing software as a suite of small modular and independently deployable applications.



CONS MONOLITHIC APPLICATION

- A nightmare to manage in large scale due to the fact that developers keeps on adding new features and changes which makes it complex and difficult to fully understand what does what
- Everything is written and coded in the same programing language
- Each deploy -> full deploy to the whole system
- Reliability and Bug hunting (Full regression anyone?)
- Change / Upgrade framework - Easier to land on the moon (again)

Microservices Benefits

- application complexity by decomposing an application into a set of manageable services which are faster to develop and much easier to understand and maintain.
- Splitting a big application in a set of smaller services also improve the fault isolation. In fact, it is more difficult for the whole system to down at the same time and a failure in processing one customer's request is less likely to affect other customer's requests.
- developers are free to choose whatever technologies make sense for their service. We are not more bound to the technologies chosen at the start of the project - Wooo Hooo!
- easier for a new developer to understand the reduced set of functionality of a microservice instead of understand a big application design

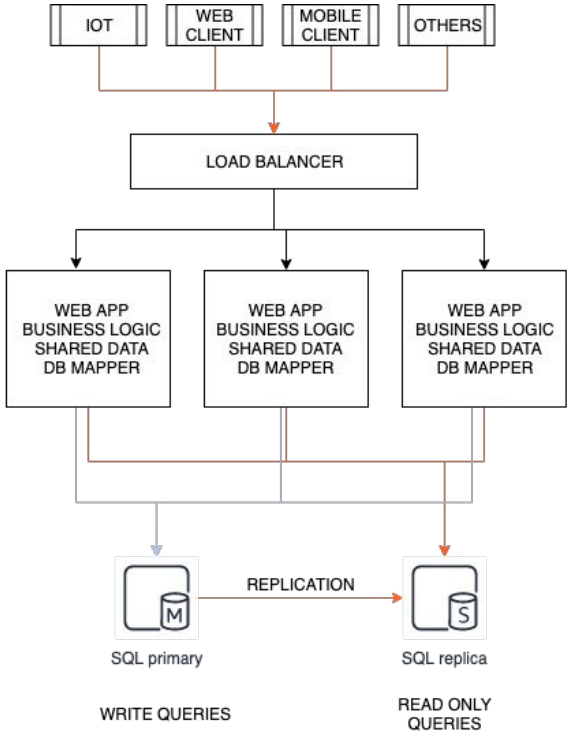
Microservices Benefits

- Each microservice could be refactored piece by piece as new technology solutions become available
- Language agnostic, so we can select the appropriate language or framework for each service
- each service can be deployed independently by a team that is focused on that service, using different technology stacks that are best suited for their purposes. This is great for continuous delivery, allowing frequent releases while keeping the rest of the system stable.
- Continuous deployment possible for complex applications and enables each service to be scaled independently.

Monolith VS Microservices: Scaling

Monolith Scaling

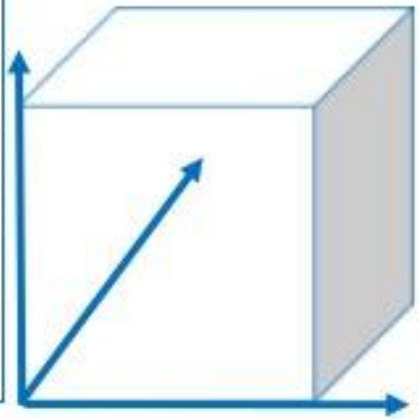
Monolith apps requires to scale the entire monolith application to achieve scalability, but what if one component requires much more resource than others?



Microservices: Scaling

Y axis

Functional Decomposition
Scale by splitting application into **multiple**, **different** services.
Each service is responsible **for one or more closely related functions**

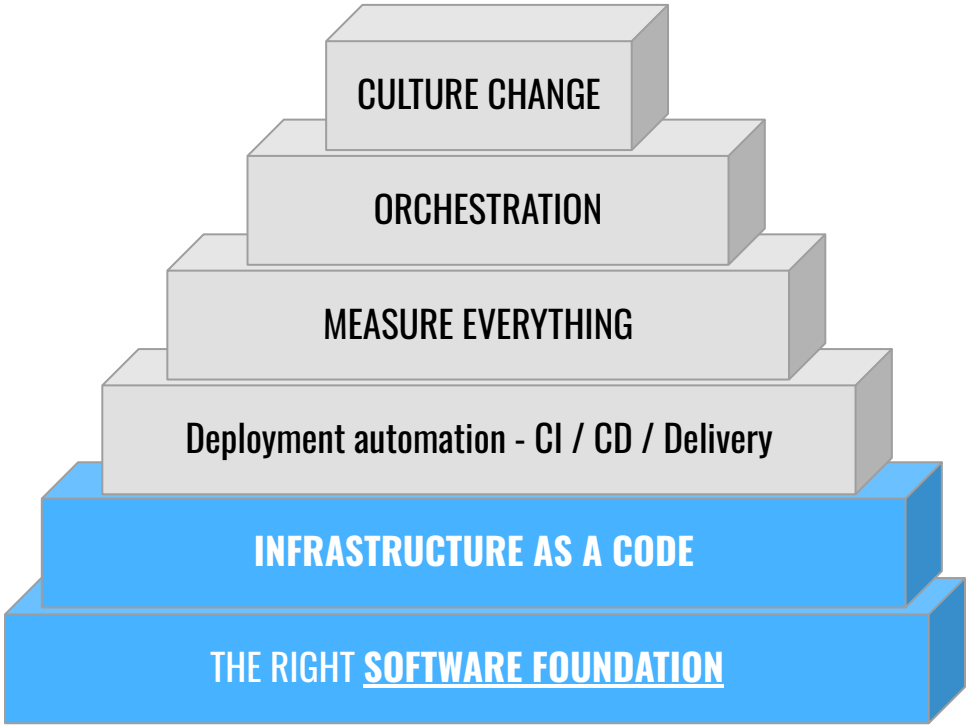


Z axis

Data Partitioning
Scale by running multiple copies of an application.
Similar to X-axis scaling but each server is responsible for **only a subset of the data**

X axis

Horizontal Scale
Scale by running multiple copies of an application behind a load balancer

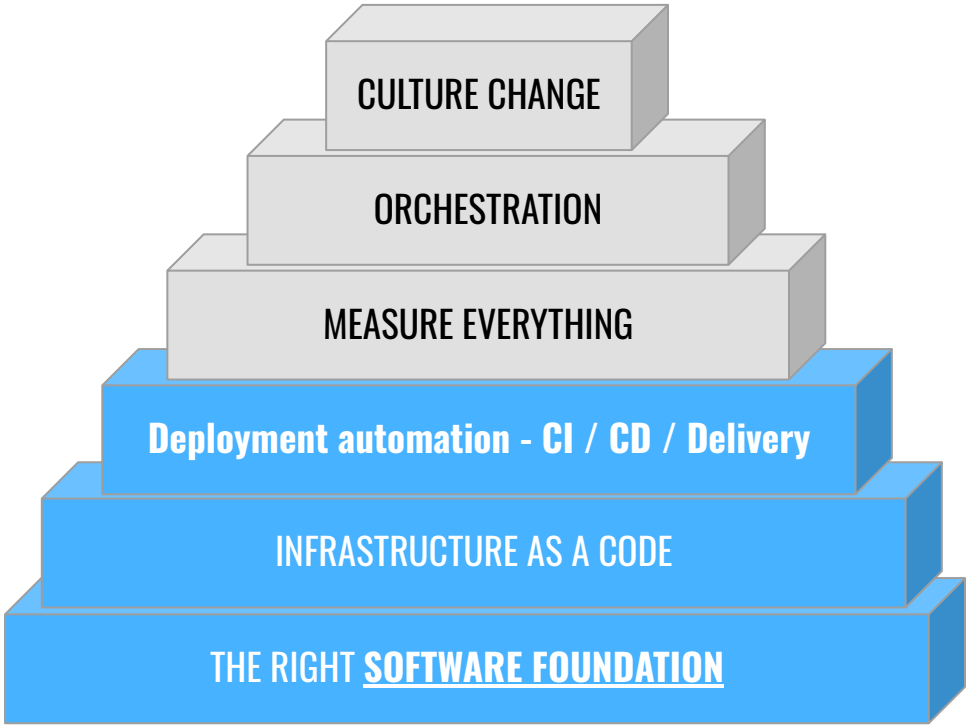

CULTURE CHANGE
ORCHESTRATION
MEASURE EVERYTHING
Deployment automation - CI / CD / Delivery
INFRASTRUCTURE AS A CODE
THE RIGHT SOFTWARE FOUNDATION

INFRASTRUCTURE AS A CODE

- Automate everything
  - Infrastructure provisioning
  - Application deployment
  - Runtime Orchestration
- Build a development Workflow
  - Write it in code
  - Validate the code
  - Source Control it
  - Test it on playground
  - Integration test it
  - Automate it’s run
  - Deploy artifact to prod

IAC AUTOMATION TOOLS

- IAC Models - AWS CloudFormation, **Terraform**, Azure ARM, Ubuntu Juju
- Hardware Provisioning - Packer, Foreman , MaaS, Crowbar...
- CM - Puppet , Chef , **Ansible** , Salt...
- Integration Testing - rspec, serverspec,
- Ansible integration tests


CULTURE CHANGE
ORCHESTRATION
MEASURE EVERYTHING
Deployment automation - CI / CD / Delivery
INFRASTRUCTURE AS A CODE
THE RIGHT SOFTWARE FOUNDATION

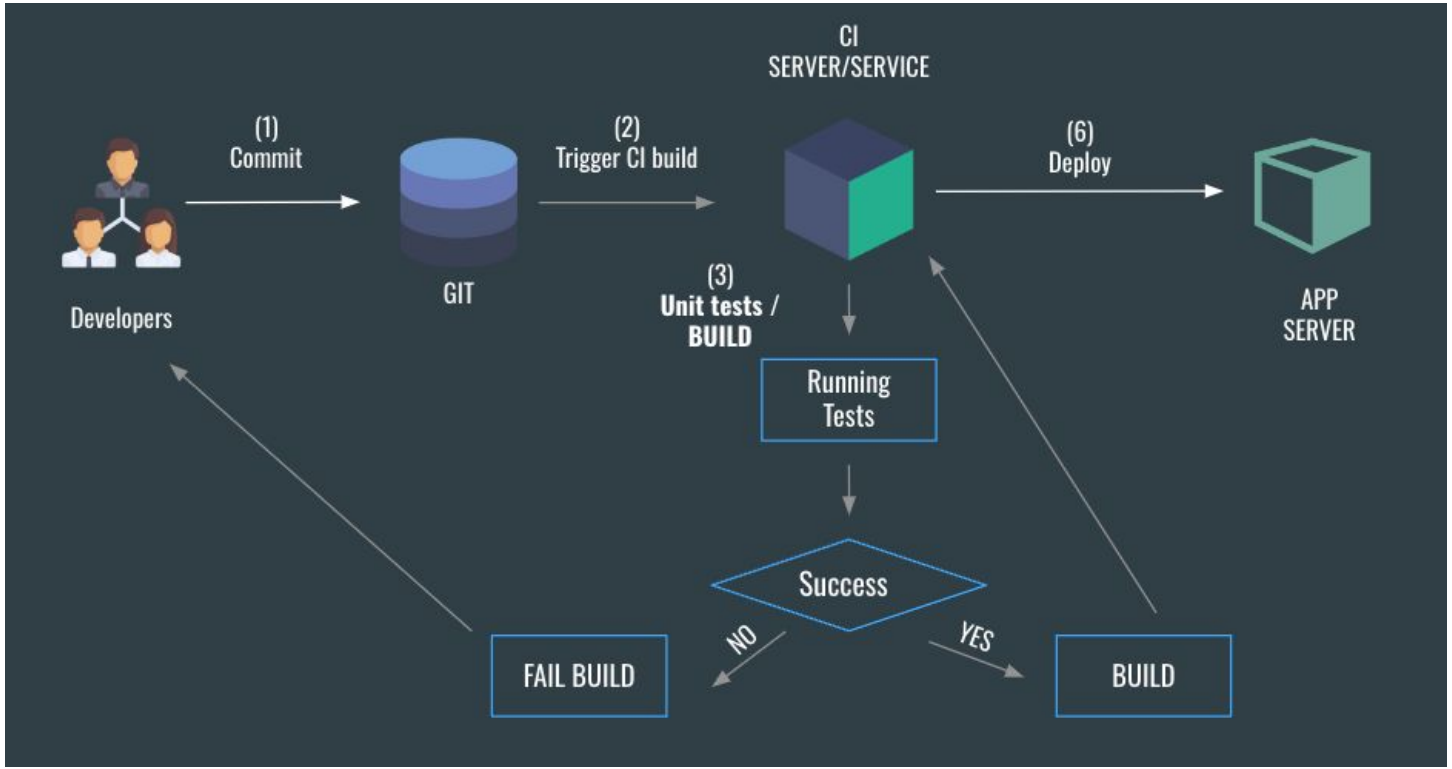
Intro

CI - Continuous Integration

Continuous Integration (CI) is a software development practice that is based on a frequent integration of the code into a shared repository. Each check-in is then verified by an automated build.

Main goal of continuous integration is to identify the problems that may occur during the development process earlier and more easily. If you integrate regularly — there is much less to check while looking for errors. That results in less time spent for debugging and more time for adding features.

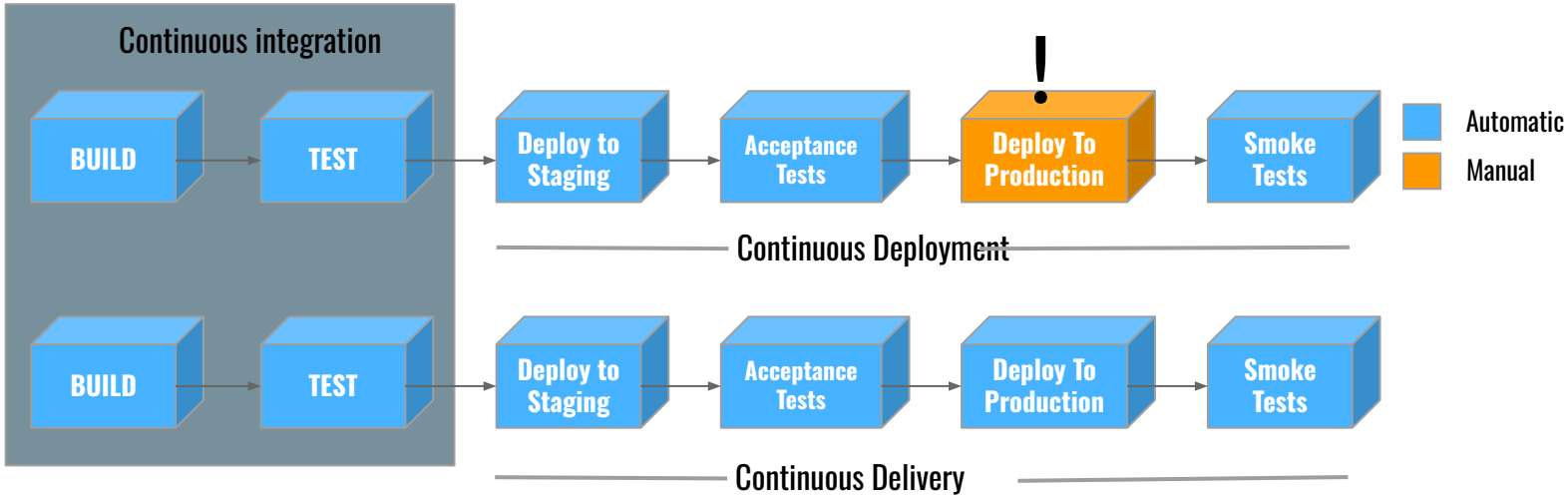
CI BASIC FLOW



CONTINUOUS DELIVERY/DEPLOYMENT

Terminology

- Integration (CI) - Build and test
- Deployment (CD) - Deploy and integration test
- Delivery - All the way to production



CI / CD DICTIONARY

Unit Tests

Is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

Static Analysis

Static analysis is one of the leading testing techniques. A static analysis tool reviews program code, searching for application coding flaws, back doors or other malicious code that could give hackers access to critical company data or customer information.

CI / CD DICTIONARY

Integration Tests

Is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

Load Tests

Is performed to determine a system's behavior under both normal and anticipated peak load conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation.

NAYA College

CI / CD DICTIONARY

Security Tests

Is a process intended to reveal flaws in the security mechanisms of an information system that protect data and maintain functionality as intended.

Acceptance Tests

Is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

Blue Green Deployment

Is a technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.

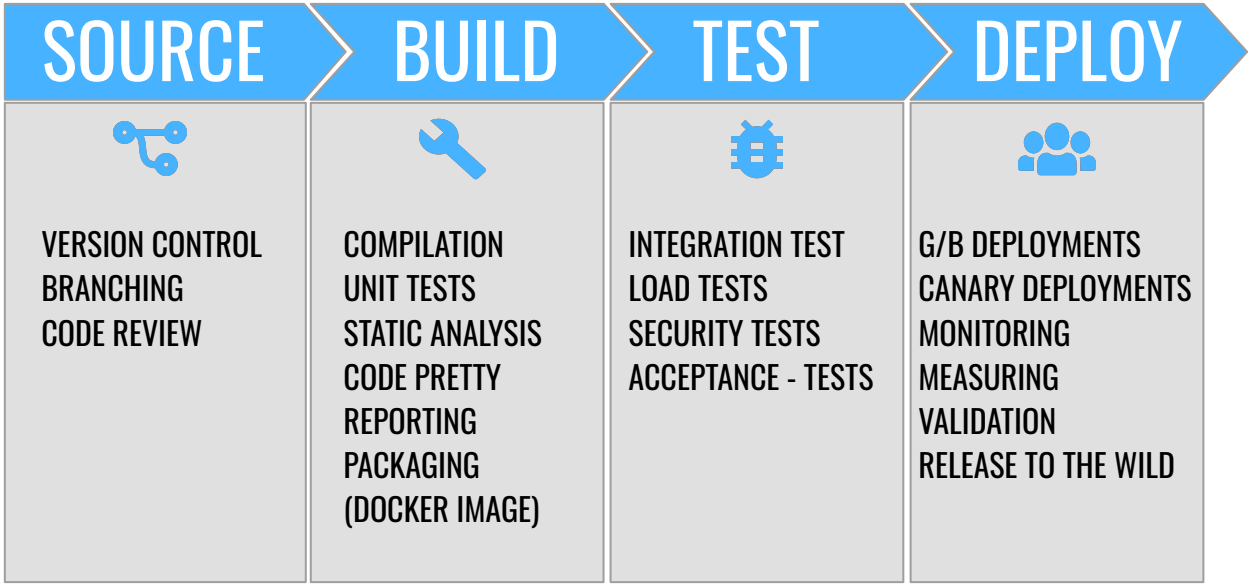
Canary Deployment

Is a pattern for rolling out releases to a subset of users or servers. The idea is to first deploy the change to a small subset of servers, test it, and then roll the change out to the rest of the servers.

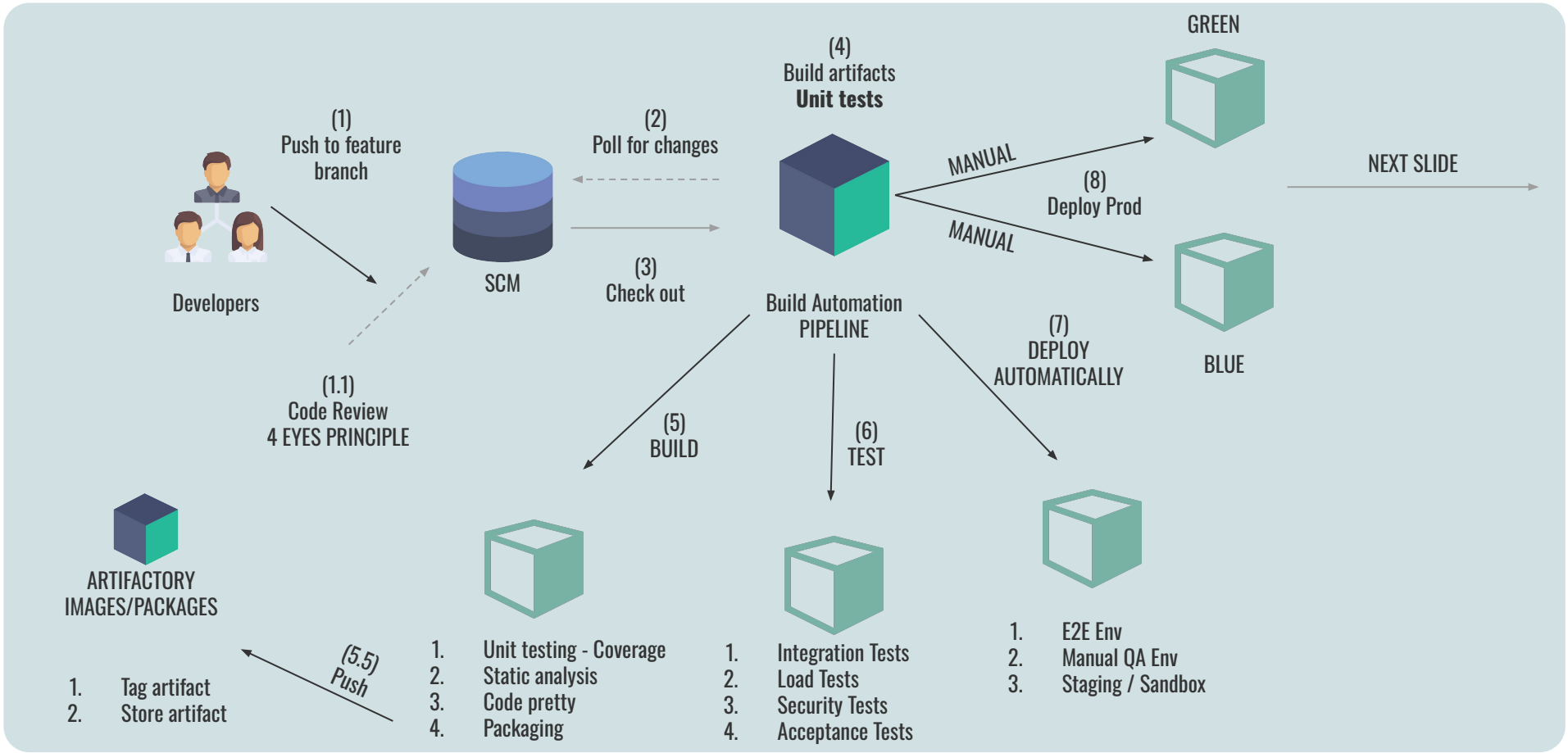
NAYA College

PIPELINE - CONTAINER BASE

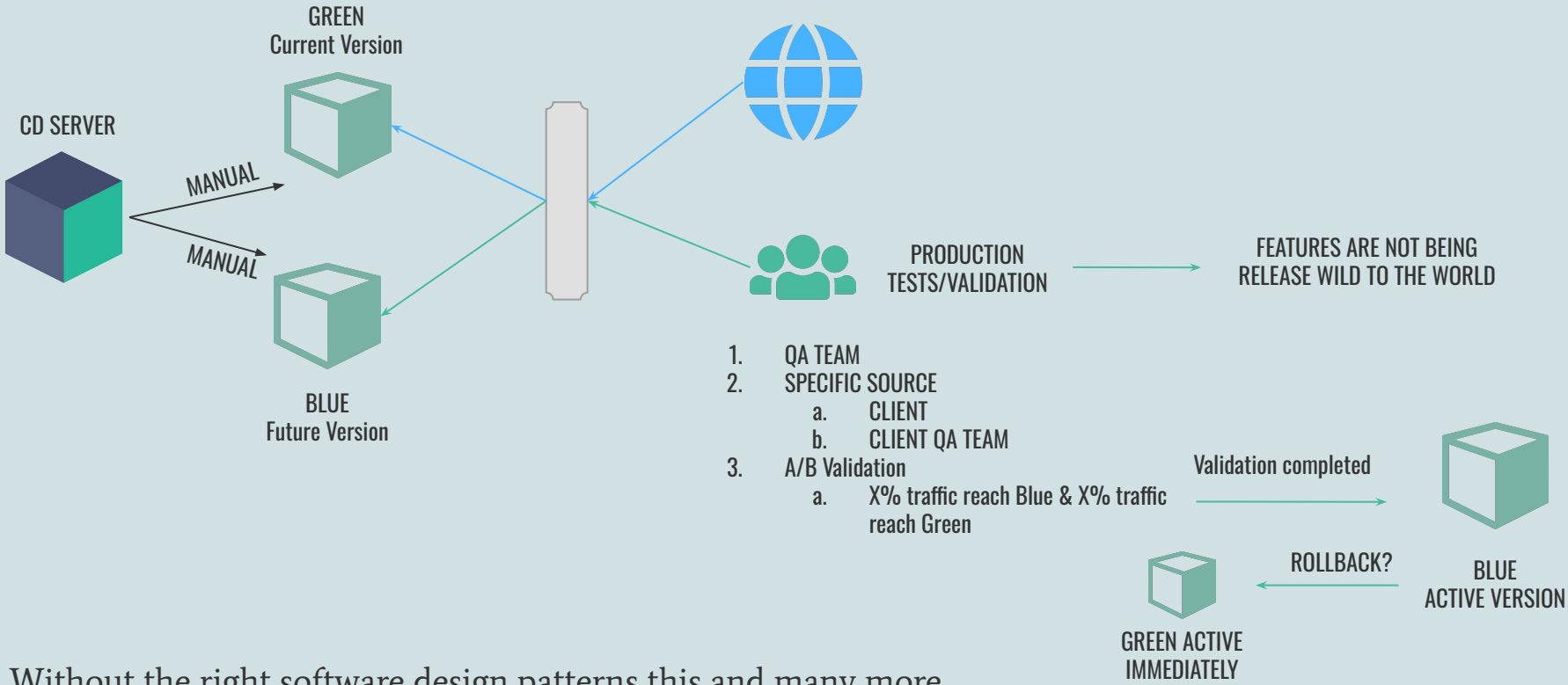
PIPELINE - CONTAINER BASE



CI/CD PIPELINE TBD

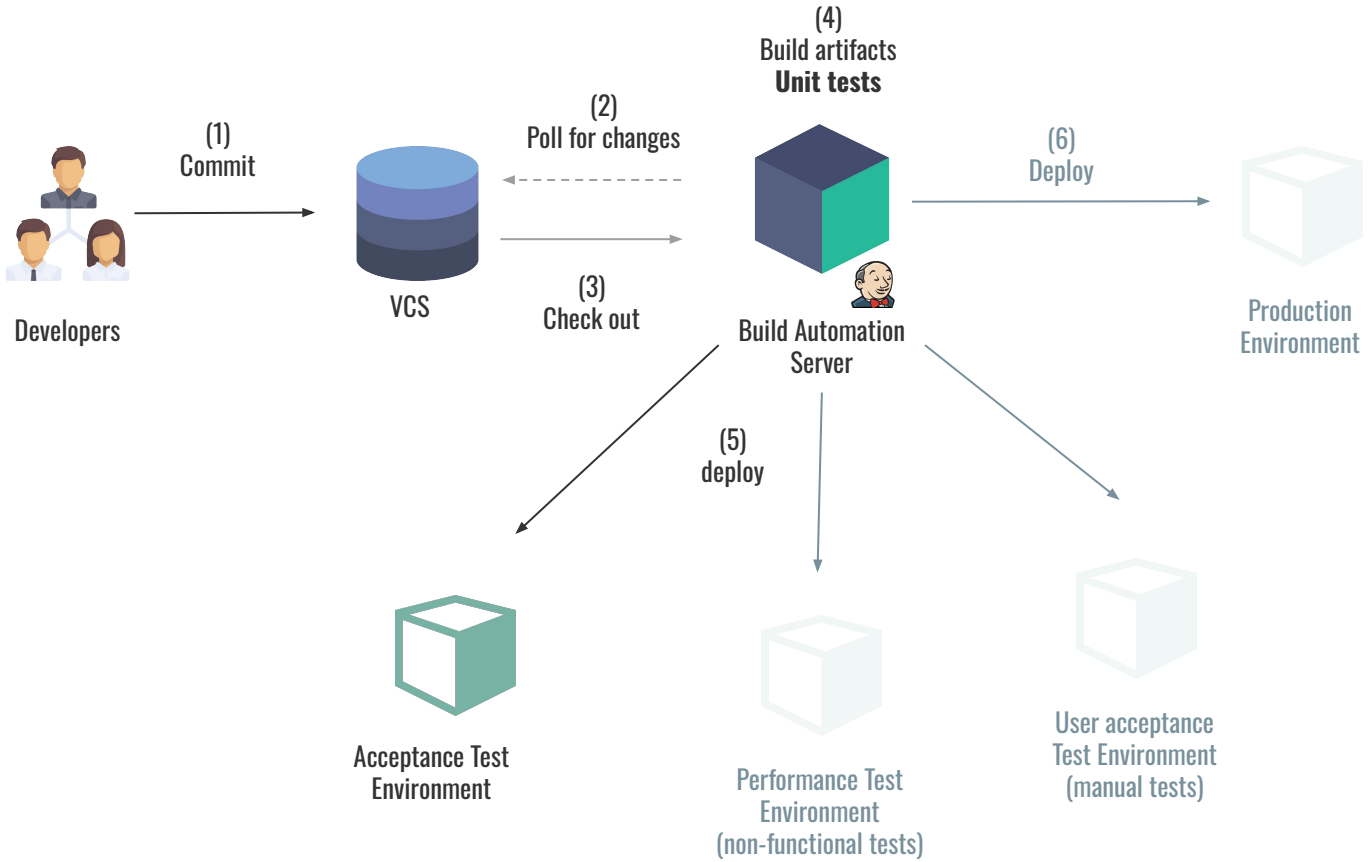


CI/CD PIPELINE TBD

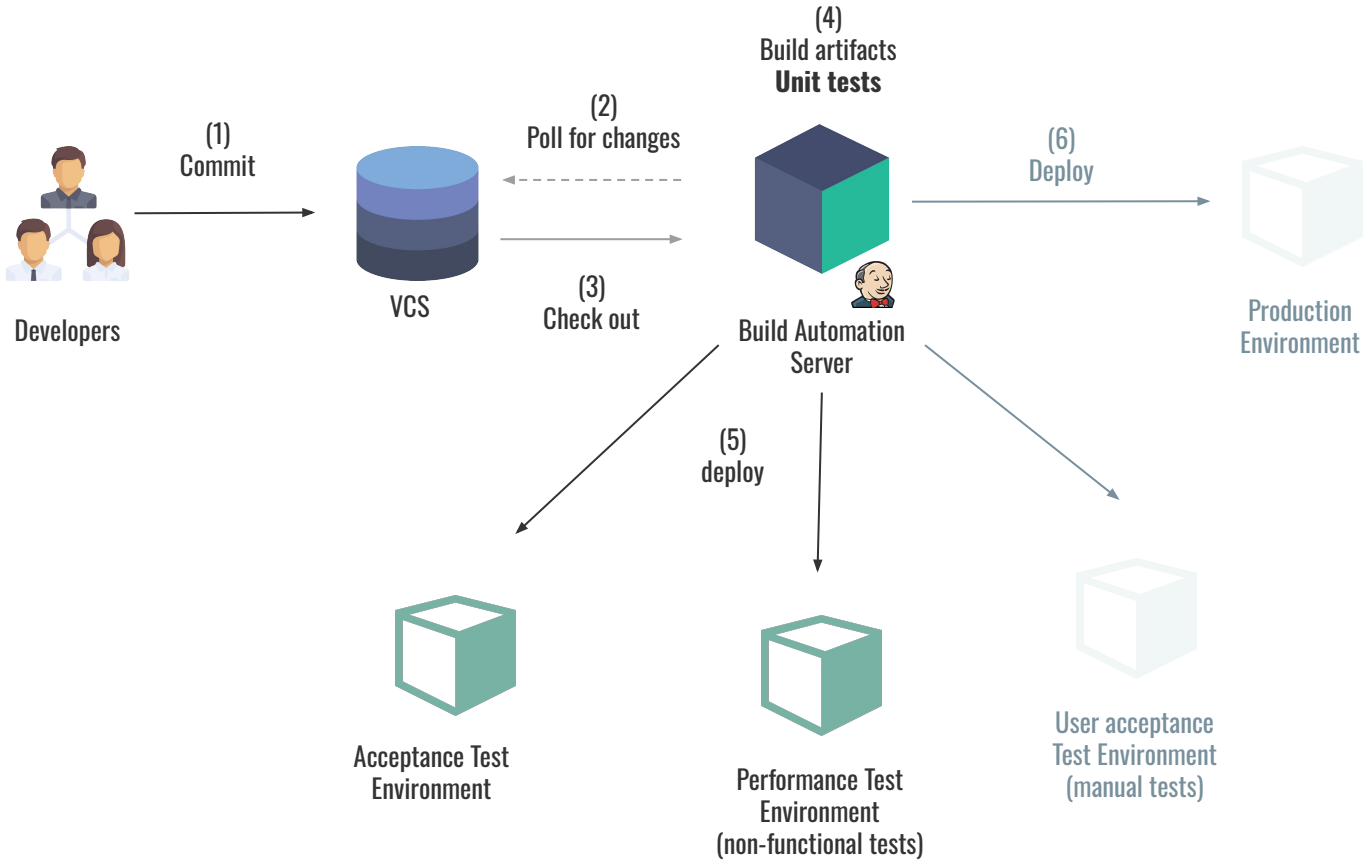


** Without the right software design patterns this and many more functionality such as rolling updates will not be achievable

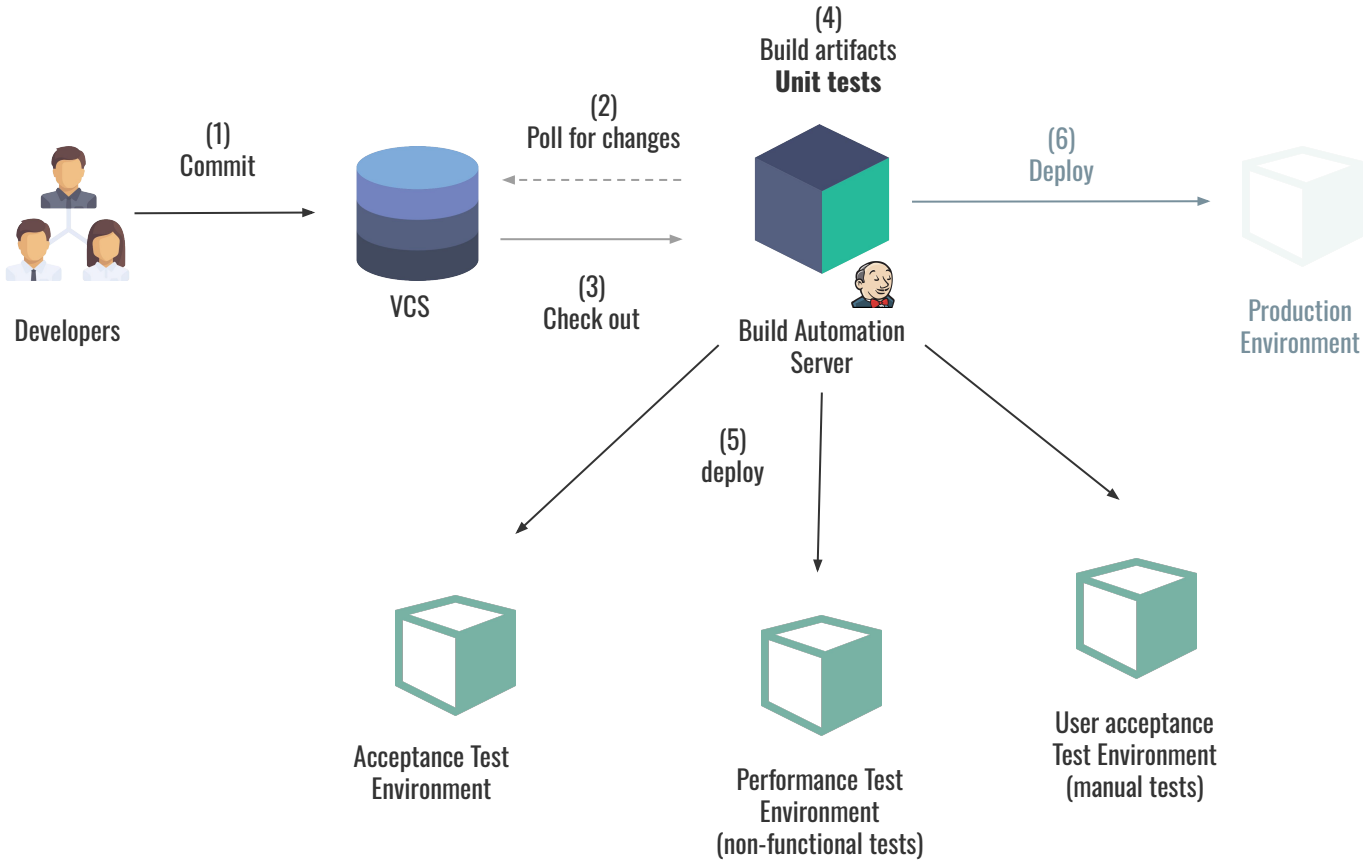
CI/CD PIPELINE



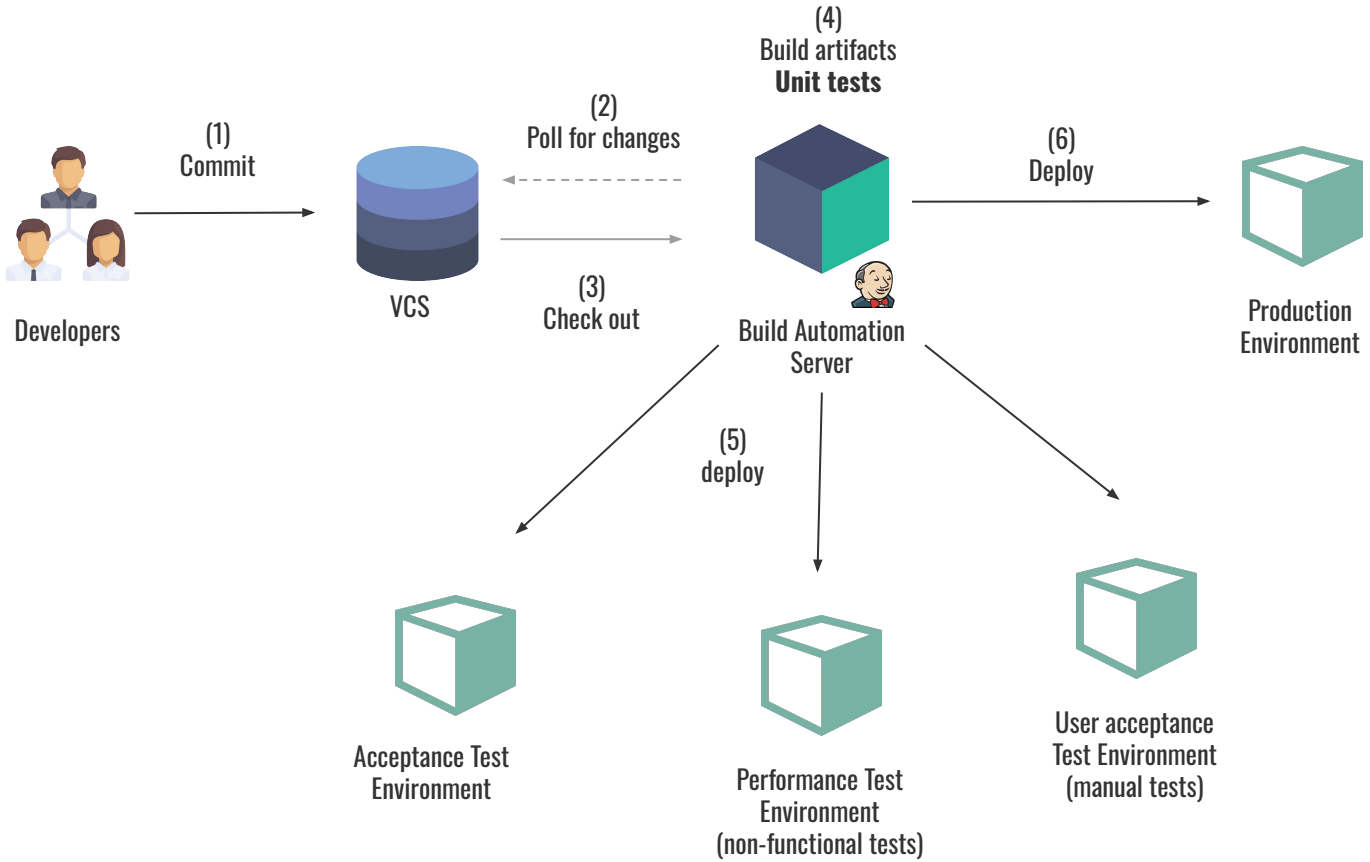
CI/CD PIPELINE



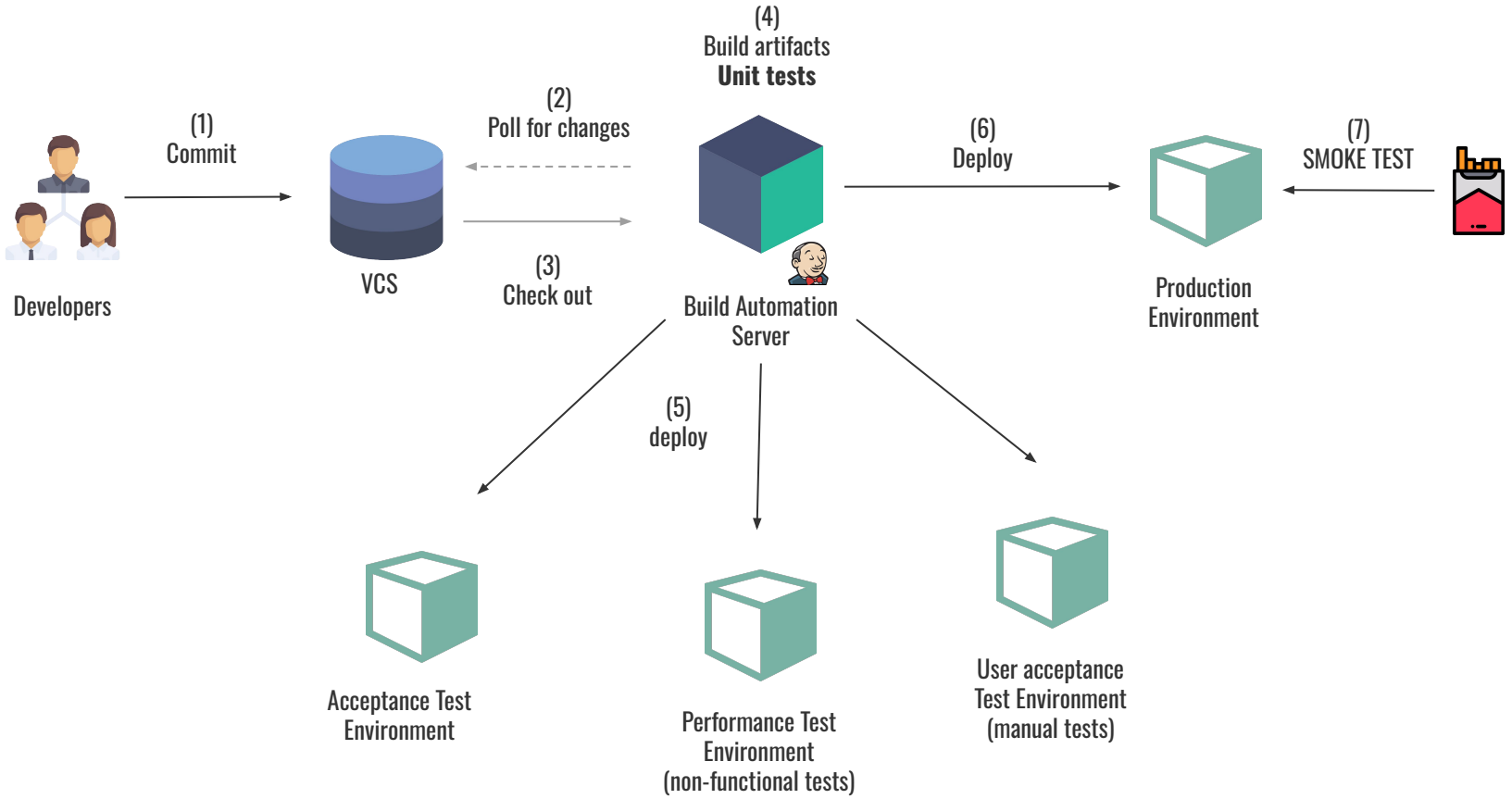
CI/CD PIPELINE



CI/CD PIPELINE



CI/CD PIPELINE



Benefits:

Automation

Reduces the risks of software releases errors that are made by manual process and environments that got deferred due to -

- Missing OS Environment variables configuration such as :
  - Max. open files
  - Users
  - Files and folders etc.
- OS Configuration and packages
- Middleware configuration and versioning.

Benefits:

Auditability

Know who did what, why he did it and when

Achieved that by

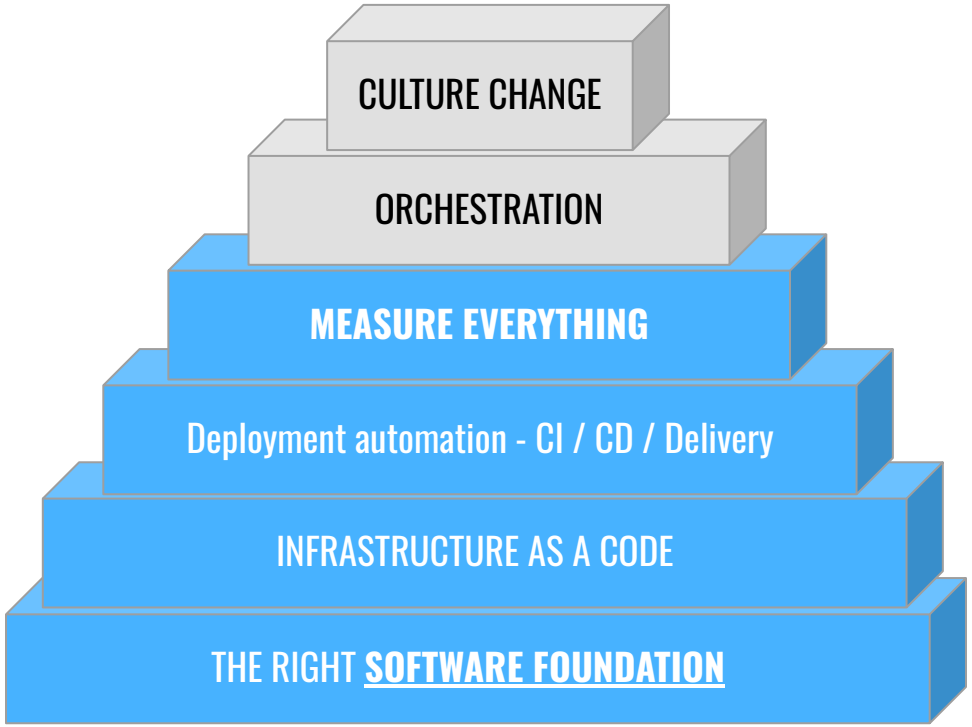
- Save all specifications and configuration in svn
- Provide meaningful information per commit by using the changelog messages
- Recreate environment base on revisions that got deployed

Benefits:

Repeatability

Deployment are no longer in the hand of one profissional

- Allows any authorized personnel to recreate environments
- Bringing “Deployment as a service” to the developers and operation team
- Catching bugs early in the flow
    - Redeploying environments in minutes with working versions - Minimizing **MTTR — Mean time to repair**
- Making deployments boring and ultra reliable


CULTURE CHANGE
ORCHESTRATION
MEASURE EVERYTHING
Deployment automation - CI / CD / Delivery
INFRASTRUCTURE AS A CODE
THE RIGHT SOFTWARE FOUNDATION

NAYA College

MONITORING ARCHITECTURE

The 5 “design” principles of monitoring cloud native apps and microservices

#1 MONITOR CONTAINERS INTERNALS

The building blocks of microservices -> containers are black boxes that span the developer laptop to the cloud. Without real visibility into containers functionality, it's very hard to perform basic functions like monitoring or troubleshooting a service. We need to know what's running in the container, how the application and code are performing, and if they're generating important custom metrics.

Than once it scales up, possibly running Hundreds of docker hosts with thousands of containers, deployments can get expensive and become an orchestration nightmare. To get container monitoring right, we have a few choices: **Developers** instrument their code directly, run sidecar containers, OR leverage universal kernel-level instrumentation to see all application and container activity. Each approach has advantages and drawbacks. **The main point is that the old methods that worked with static workloads on VMs is just not going to cut it anymore.**

#2 USE ORCHESTRATION SYSTEMS

One of the most critical processes we need to perform is tracking aggregate information from all the containers associated with a function or a service, like what is the response time of each service. This kind of on-the-fly aggregation also applies to infrastructure-level monitoring to know which services / containers are using resources beyond their allocated CPU shares.

#3 PREPARE FOR ELASTICITY (UP/DOWN)

PREPARE FOR ELASTICITY (UP/DOWN)

Container-native environments change quickly, and that sort of elasticity exposes the weaknesses in any monitoring system. **Manually defining and tuning metrics may work well for 20 or 30 containers, but microservice monitoring must be able to expand and contract alongside elastic services—and without human intervention.**

Meaning that if we must manually define what service a container is included in for monitoring, we are likely to miss new containers spun up during the day by K8S. In the same vein, if installs a custom stats endpoint when new code is built and deployed, monitoring is complicated as developers pull base images from a Docker registry.

#3 PREPARE FOR ELASTICITY (UP/DOWN)

PREPARE FOR ELASTICITY (UP/DOWN)

This is the main reason why systems such as Nagios, Zabbix, Sensu and other very good monitoring tools no longer fits (natively),

As they require manual intervention when new endpoints / services / tasks are being added.

monitoring Framework was built on elasticity and data aggregation from multiple sources in mind such as:

- AWS CloudWatch
- Containers internals
- K8S Metadata
- Endpoints such as - RDS, ALBs. ASG and many many more
- 3rd party services such as pingdom
- Last but most important - Application metrics (Developers made)

NAYA College

#4 MONITOR API

As the “native language” of microservice environments, APIs are the only elements of a service **exposed to other teams/services/roles.** An API’s response and consistency may even become the internal service level agreement if a formal SLA hasn’t been defined. That means API monitoring must go beyond the standard, binary up-down checks.

As a user of microservices we will find it very valuable to understand our most frequently used endpoints as a function of time, letting us see what’s changed in services usage. whether because of a design or user change. Discovering the slowest service endpoints can also show us areas that need optimization or to be scale UP/DOWN automatically using KPI monitoring. Being able to trace service calls through the system, a function typically used only by developers, will help our monitoring framework to better understand the overall user experience. That aspect of API monitoring will also break down information into infrastructure- and application-based views of the microservices environment.

#4 MONITOR API

Once those type of KPI’s will be exposed and available for our monitoring to use -

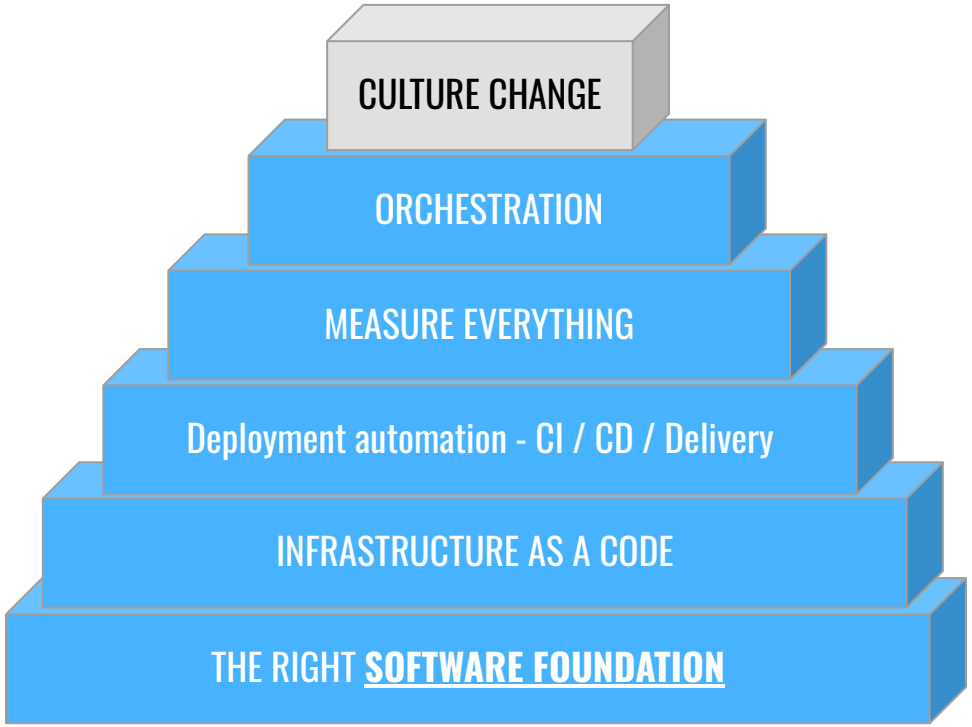
We will be able to build smart auto scaling mechanism and an alert system that notify and direct us to the exact location of the issue.

#5 MAP MONITORING TO ORGANIZATIONAL STRUCTURE

While microservices presage a comprehensive shift in how we **monitor** and **secure** our software infrastructure, it's essential that we will not overlook the people aspects of software monitoring. We need to make sure our monitoring platform allows each microservice personal to isolate its alerts, metrics, and dashboards, while still giving operations a global view across the system.

With this final point we should understand why the following slides will describe multiple self dependent (and decoupled by design) but Complement each other to provide -

- Time series dashboards and monitoring
- Active and Passive checks for native cloud design and microservices
- Biz Dashboard with capability to be built by anyone and creates alerts as One required and base on the "data" that interests him.

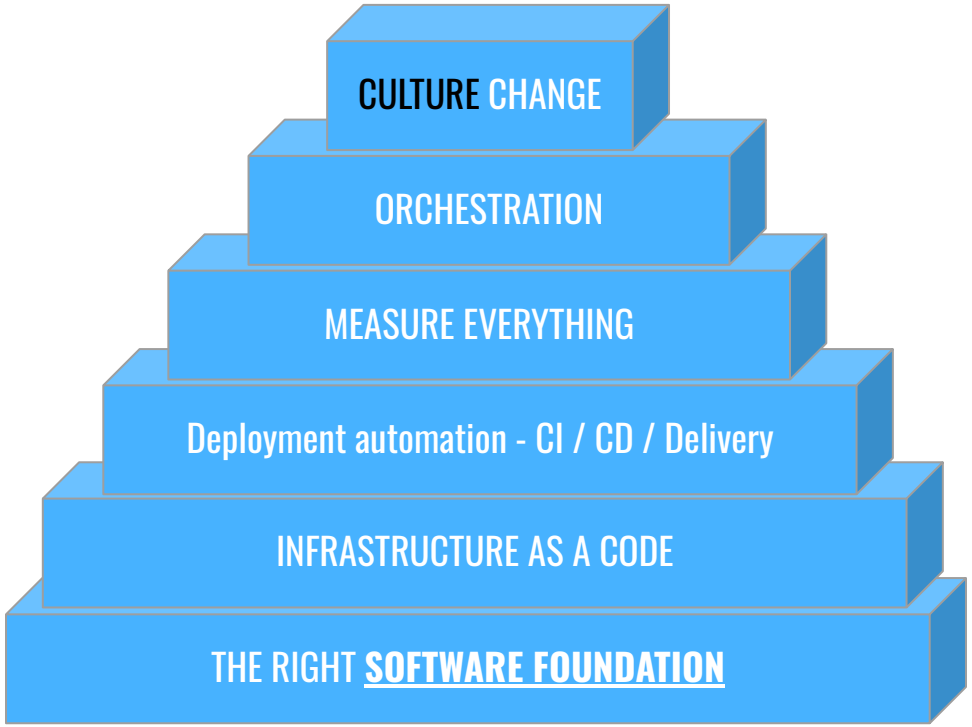

CULTURE CHANGE
ORCHESTRATION
MEASURE EVERYTHING
Deployment automation - CI / CD / Delivery
INFRASTRUCTURE AS A CODE
THE RIGHT SOFTWARE FOUNDATION

NAYA College

- MICROSERVICES? - CONTAINERS THEN

Docker is an open platform for developing, shipping, and running applications.

Docker allows you to package an application with all of its dependencies into a standardized unit for software development.


CULTURE CHANGE
ORCHESTRATION
MEASURE EVERYTHING
Deployment automation - CI / CD / Delivery
INFRASTRUCTURE AS A CODE
THE RIGHT SOFTWARE FOUNDATION

Culture - Let’s briefly check Accelerate State of DevOps 2023

Q&A