DevOps

Enablement Workshop

Agenda

- What is DevOps?
- Why DevOps?
- DevOps – VCS & GIT
- DevOps – Ansible
- DevOps - CT
- DevOps - Containerisation
- DevOps - Terraform
- Orchestrate Containers - Kubernetes
- DevOps – Application Delivery (CI & CD)

Atgen

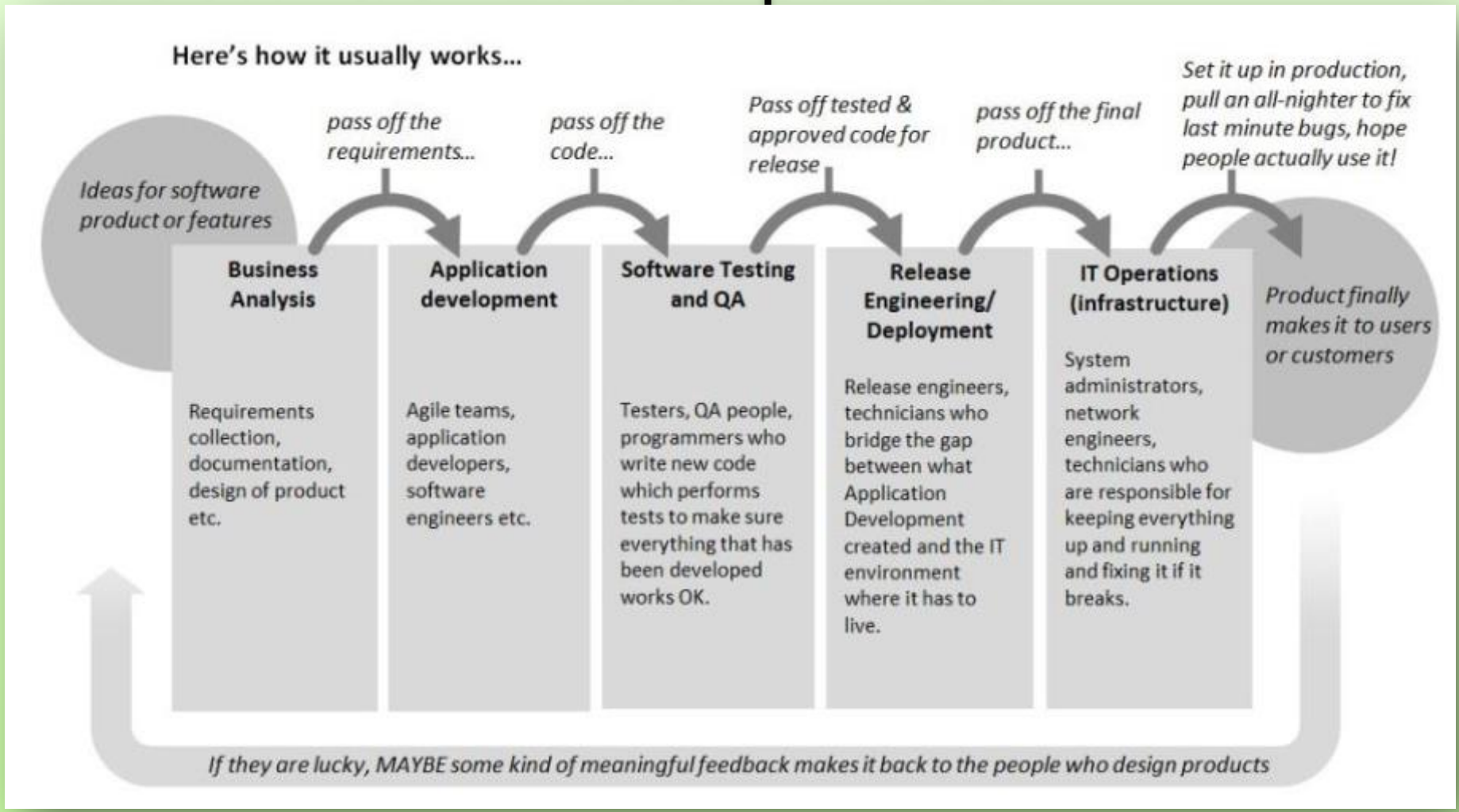
Some Assumptions

- DevOps, by definition, is never a “one-size-fits-all” remedy
- DevOps is about drilling down on your own organization’s specific problems and challenges

Session: 1

What is DevOps?

A look at Enterprise



Atgen

What is DevOps?

"DevOps is development and operations collaboration"

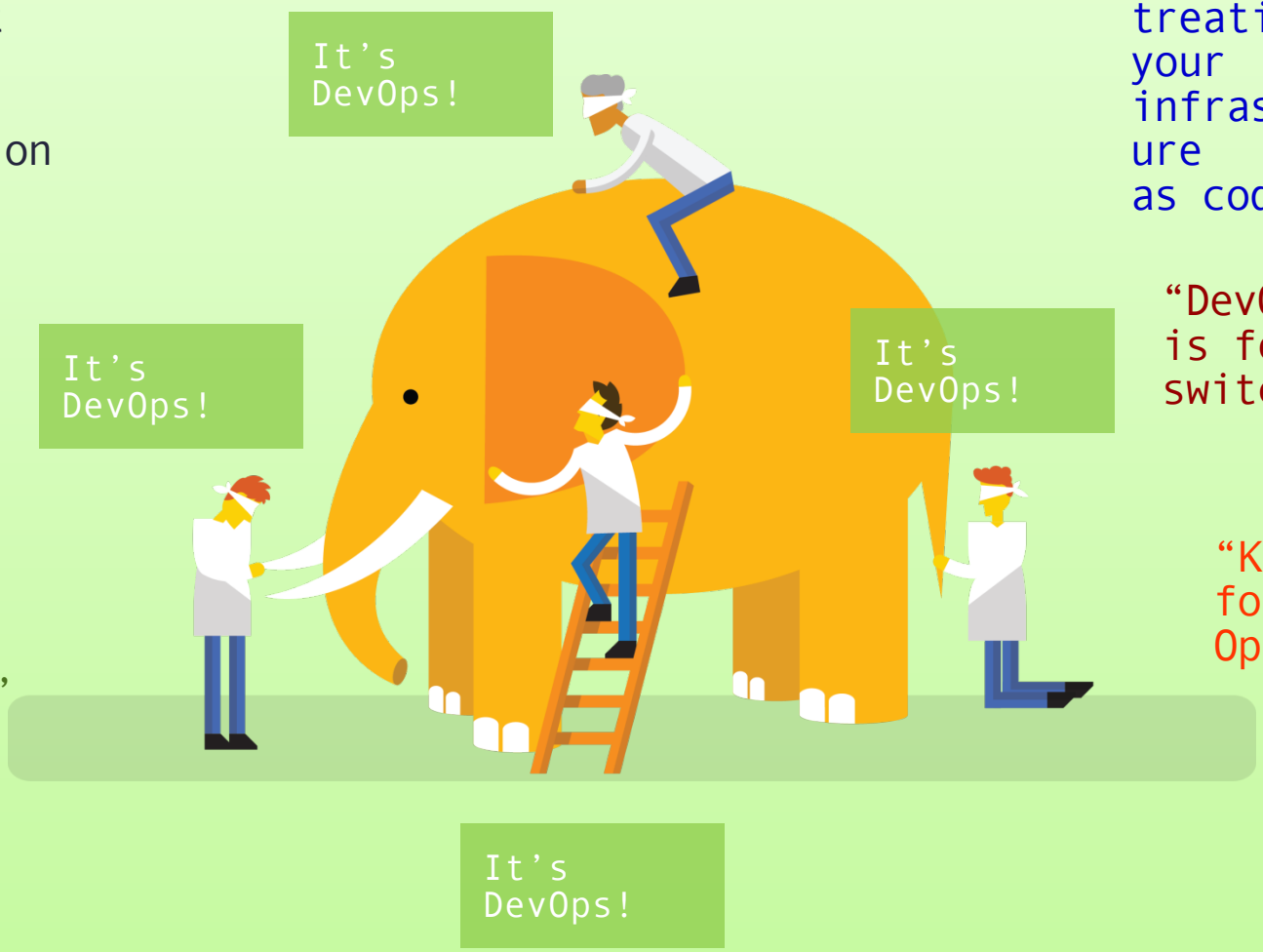
"DevOps is treating your infrastructure as code"

"DevOps is using automation"

"DevOps is feature switches"

"DevOps is small deployments"

"Kanban for Ops?"



Atgen

Welcome to the 21st century!

Helps design requirements not just for the end product, but also for automated tests that will be used along the way. Same with infrastructure requirements

Still develops the code for the product, but responsibility now no longer ends when their coding is done. They have to work with the team to make sure what they built works...all the way out to production.

Is now no longer a separate department from development, but works with them to set up tests as automated "quality gates" that allow developers to check their work automatically and get immediate feedback if something is broken so they can fix it

Now no longer the "final step" for a product – they become part of the team from the start and act as technical support for the developers to make sure infrastructure matches needs and is the same during development as it will be in production

Now everybody has access to everything everyone else is using.

Development code

All the software code is stored in a common location. A version control tool keeps everything in sync for everybody.

Testing code

Instead of waiting to build a test later, in a separate team, tests are built as you go, bit by bit – by the same teams. Tests are automated and stored in a common location, also kept in version control just like the main product is.

Infrastructure code

The infrastructure (computers, servers, databases, etc.) is all managed as virtual machines. This allows all the infrastructure to be kept in a common location and kept under version control, just like everything else.

Atgen

What is DevOps

CALMS

- Culture
- Automation
- Lean
- Measurement
- Sharing

Atgen

What is DevOps

HIGH TRUST PERFORMANCE

- Unified mission
- Aligned incentives
- Little fear/ failure/blame
- High quality of work life

Atgen

What is DevOps

IT capabilities = strategic assets, not cost centers

- Projects, features and work flow through fast cycles times
- Systems are “anti-fragile”
- IT processes & capabilities are aligned with overarching organizational needs

Atgen

What is DevOps

Automated Mature Deployment pipeline

- Technical phases of projects supported by common tools and automation processes
- Collaboration replaces handoffs
- IT infrastructure is agile

Atgen

What is DevOps

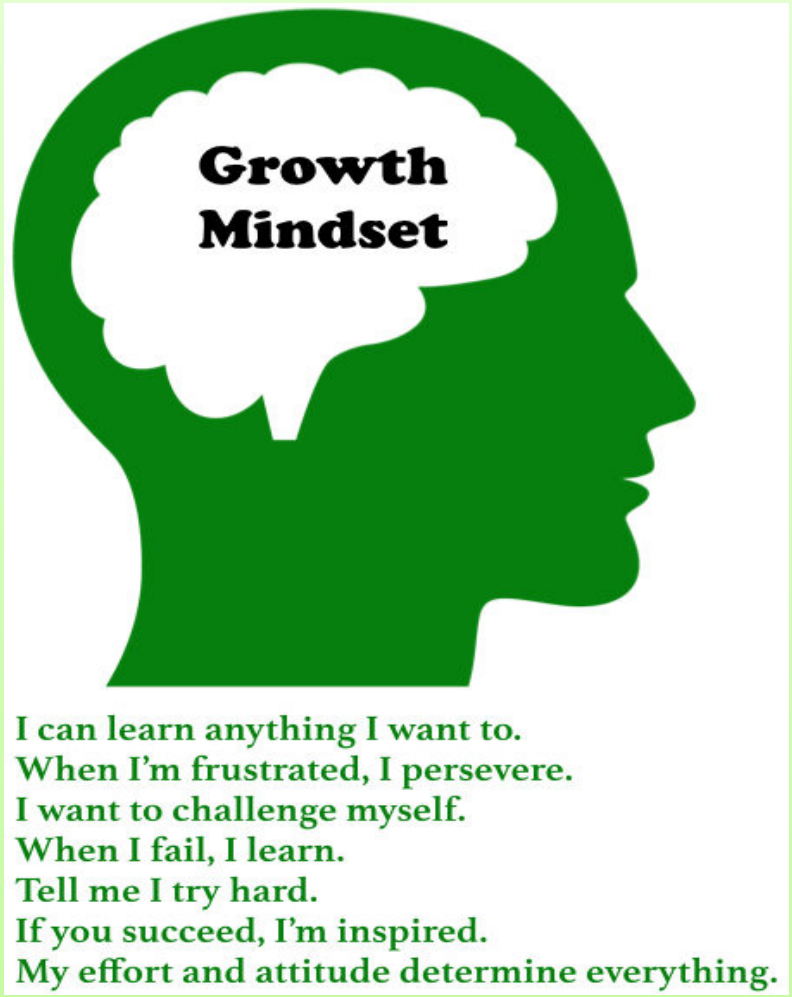
Continuous delivery of software and IT value

- Features, projects and IT work follow a regular, iterative flow
- Cycle time is short
- Workflow favors small frequent changes

Atgen

What is DevOps

Continuous Learning & improvement



- Disciplined feedback loops quickly travel back upstream for inclusion
- Tools for monitoring, measurement and alerting implemented & effective.
- Shared knowledge repositories.

Atgen

What is DevOps

Concept / ideation

Value

DevOps is not about IT problems:
DevOps is about business problems.

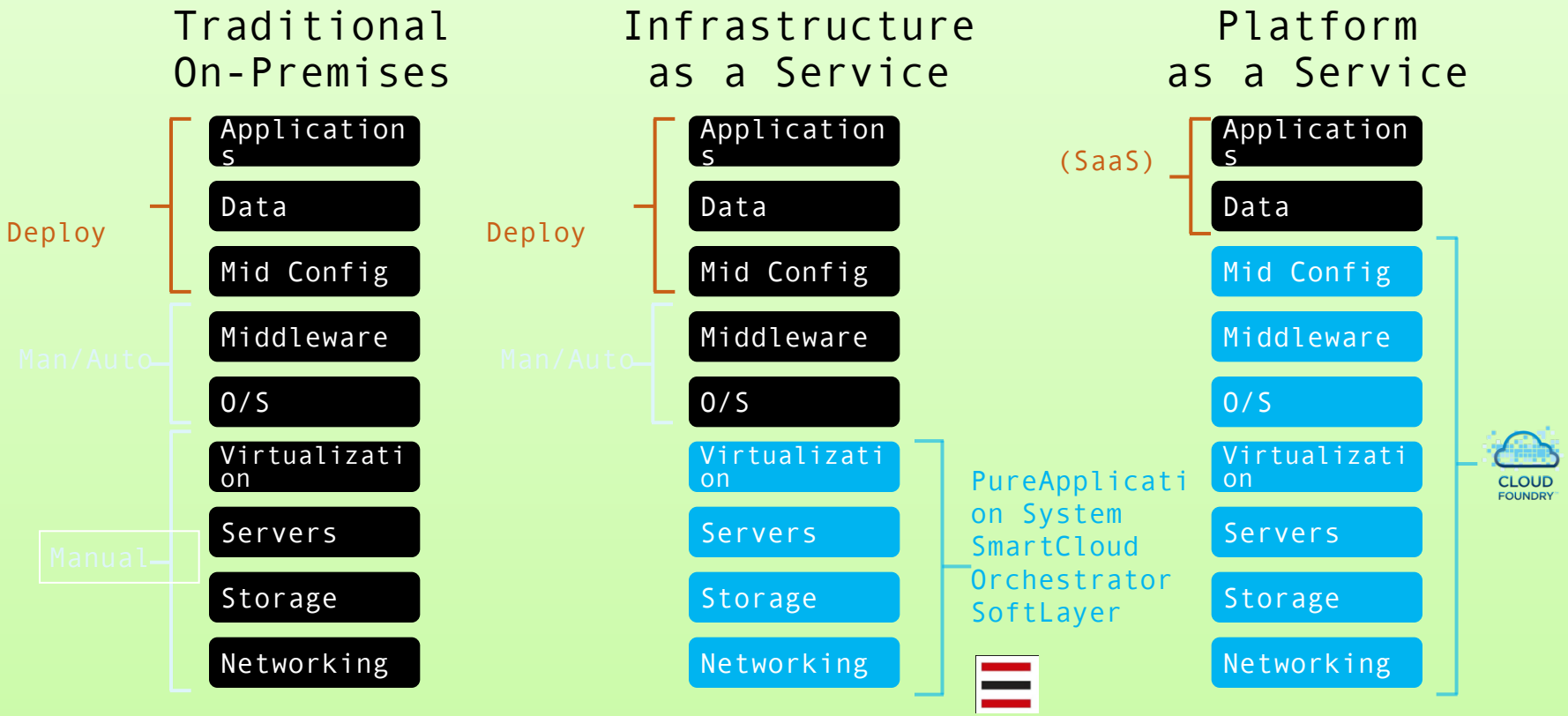
Atgen

What is DevOps?

- DevOps is culture for collaboration, Integration and Communication between different cross functional teams (including ops) for Continuous delivery.
- DevOps encourages Operations to participate early in the Development cycle so that products are designed to be easily deployable and maintainable.
- DevOps emphasizes on keeping WIP/Inventory low and go to production ASAP.

DevOps & Cloud Adoption

Automating for faster delivery with DevOps and cloud



Customization; higher costs; slower time

Standardization; lower costs; faster time

Atgen

DevOps Timeline

2007

Patrick Debois start's assessing IT Value Chain

2008

Agile System Administrators Group is launched on Google

Google

Agile2008 Conference

Andrew Clay Shafer and Patrick Debois meet at Agile Conference 2008

2009

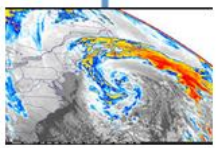
Inaugural "DevOps Days" are held in Ghent, Belgium

CHEF

Open Source toolsets rip up the legacy playbook.

John Allspaw and Paul Hammond present "10 Deploys per day" at Velocity

2010



The "Perfect Storm" of adjacent methodologies occurs

2011

Industry leading software vendors increase market presence with "Enterprise" class DevOps tools

IBM

hp

ca technologies

Industry leading software vendors increase market presence with "Enterprise" class DevOps tools

2012

Gene Kim releases "The Phoenix Project"

2013

DevOps begins to provide positive impact to "Enterprise" IT

2014

devX is born and Xceed launches the "12 days of DevOps"

2015

What does the future of DevOps hold for your organisation?

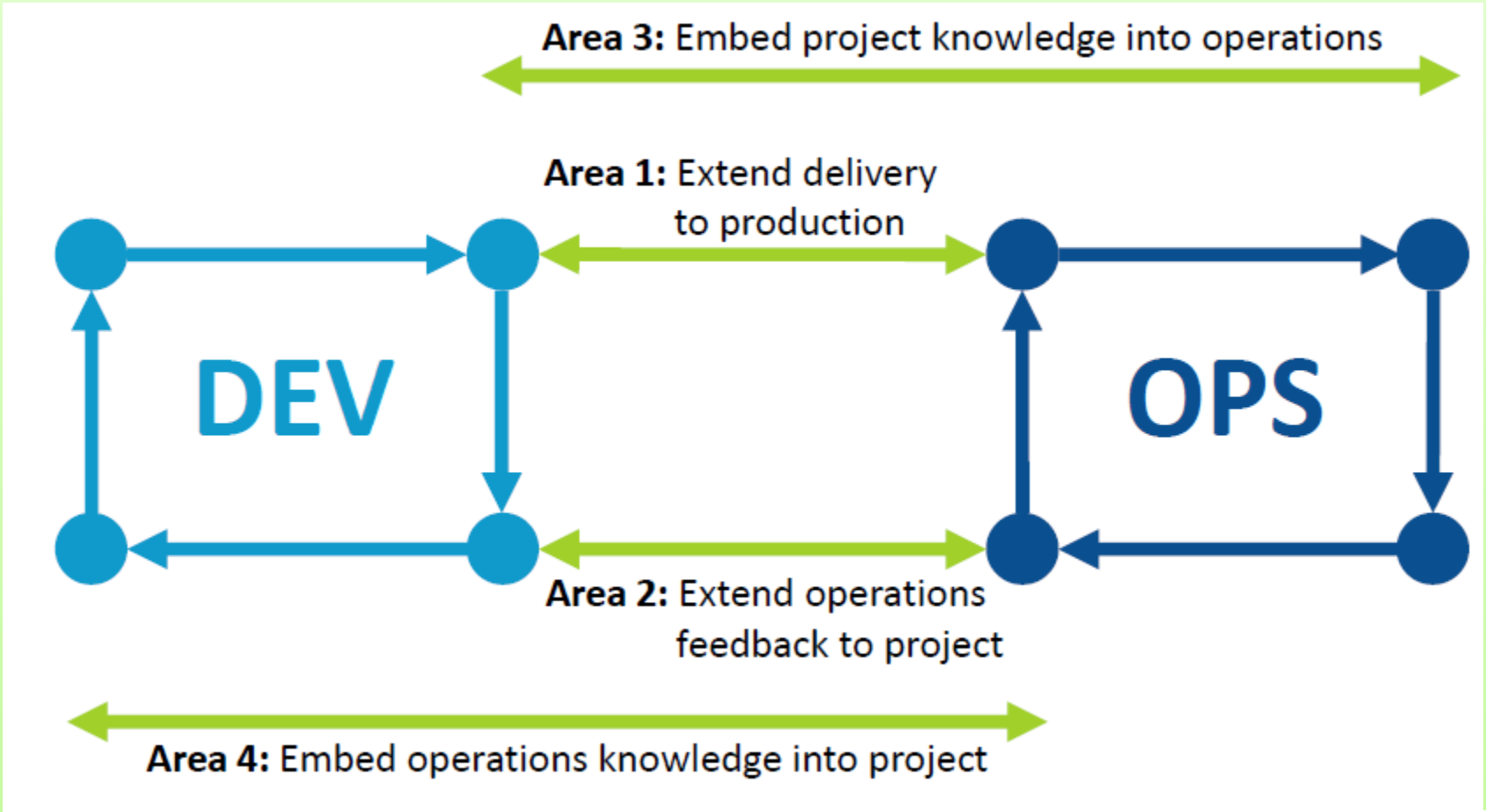
Atgen

“DevOps is a methodology with both a technological and collaborative component. DevOps is an outgrowth of the Agile method and relies on the Agile process for developing code. It also aims to improve the first steps of creating a component, such as developing stories and requirements, and the last stage of the process, which is releasing the code. One of its basic DevOps tenets is to break down the barriers between infrastructure and code and to blur or even eradicate the boundaries between development and operations.”

DevOps - Point of View

Atgen

DevOps Objectives



Atgen

Pillars of DevOps

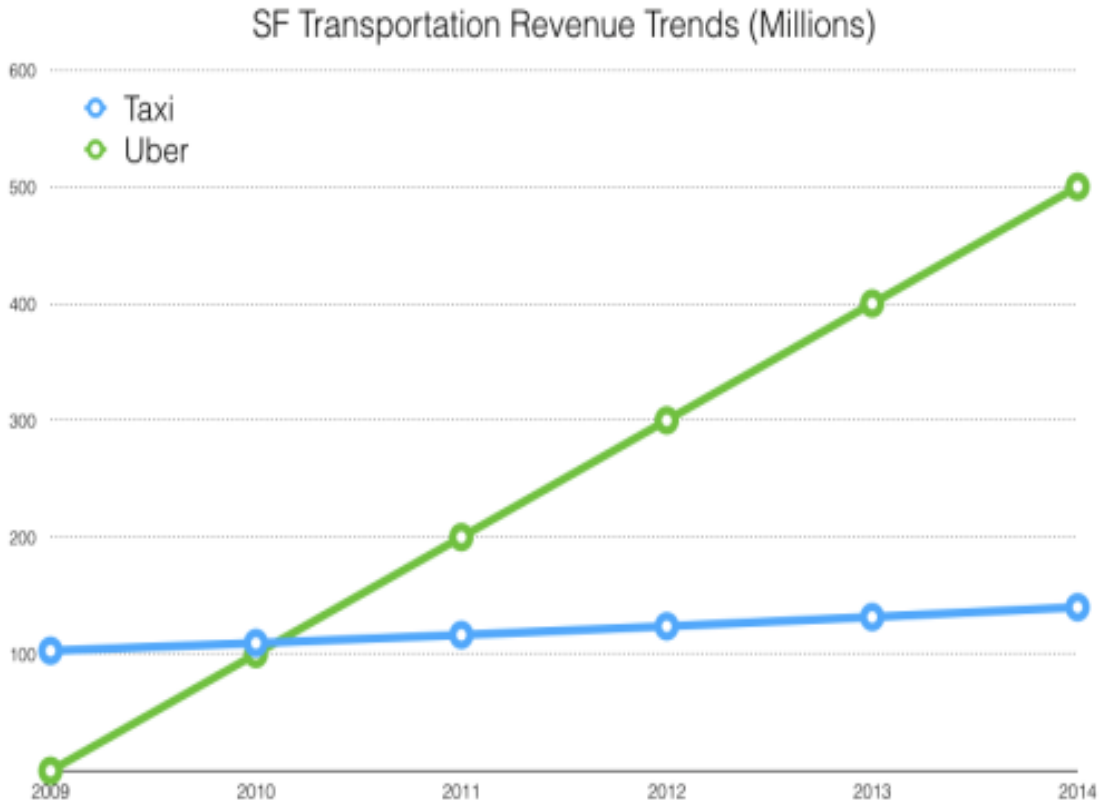
- Integration
- Collaboration
- Communication

The pace of change in our economy is accelerating remorselessly

Atgen

UBER

San Francisco taxi trips down 65% over past 15 months

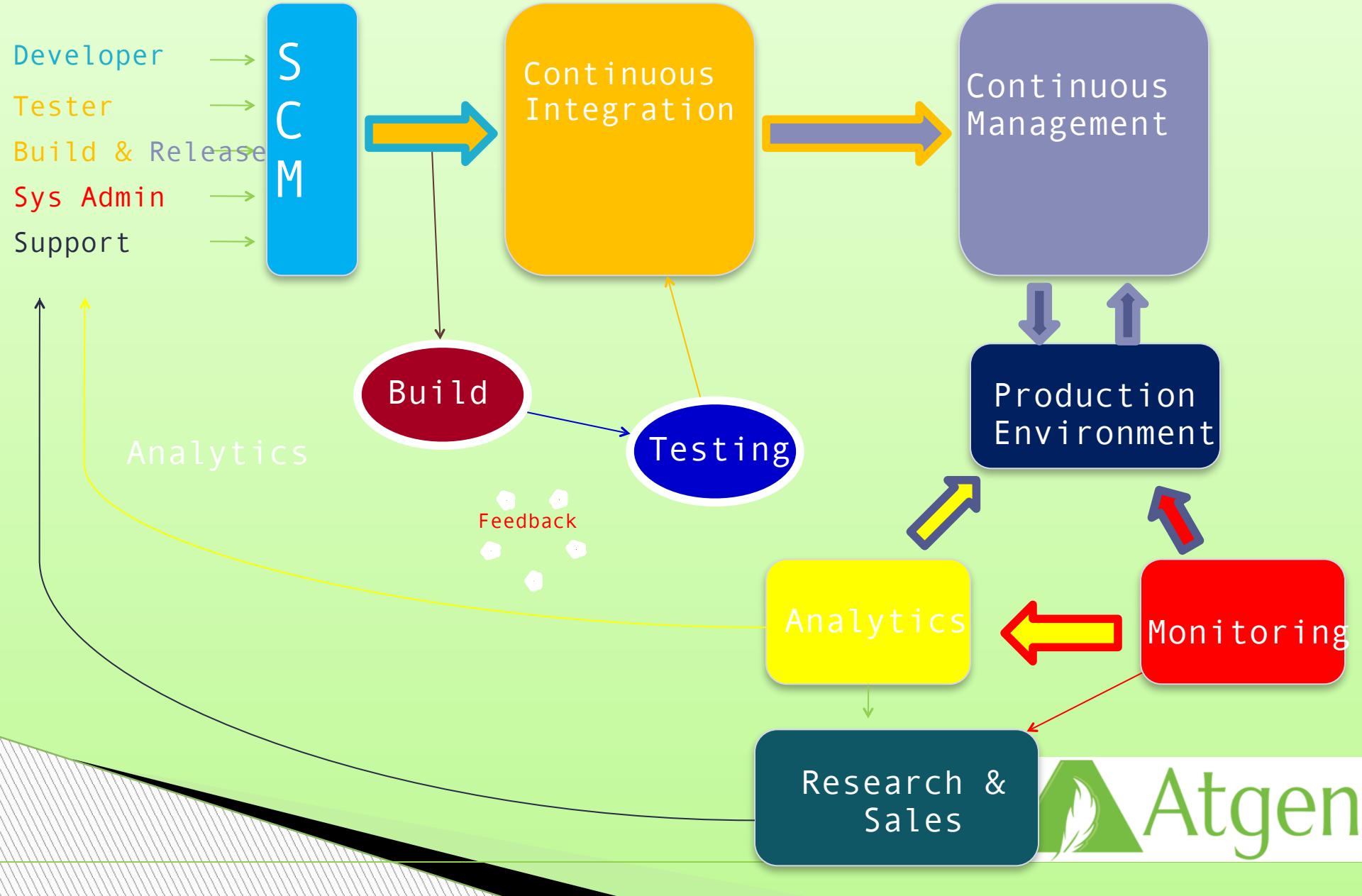


Uber 3X Taxi Revenues in 5 Years

Atgen

“There's a new trend called "devops" that is sweeping
the enterprise IT world and its become a life-or-death career
situation for many IT departments…
Like the manufacturers were in the 1970s and 1980s were
fighting for their lives,
today's IT departments are going to fight for their survival.”

What is DevOps?



Steps to achieve DevOps

- Configuration Management
- Continuous Integration
- Automated Testing
- Metrics
- Collaboration
- Making smart use of smart people

Session: 2

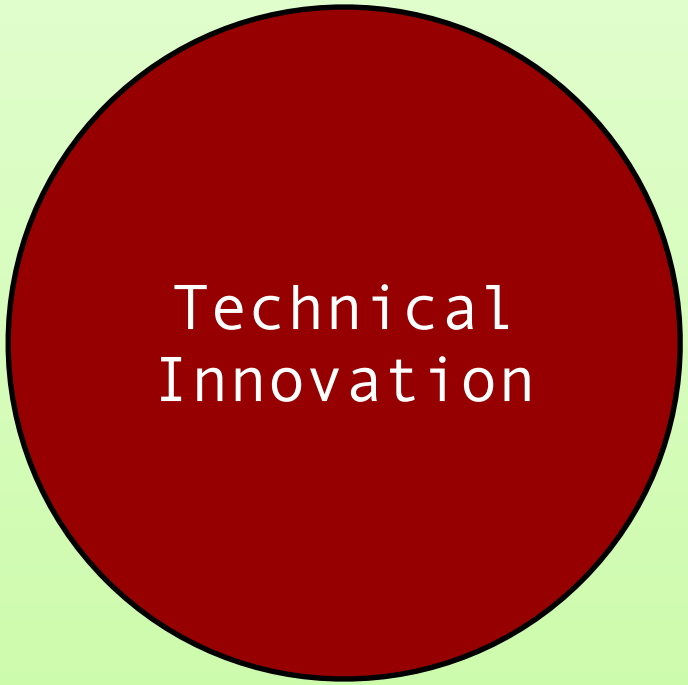
Why DevOps?

DevOps – Business Agility

- Time-to-Market Acceleration
- Experimentation
- Rapid Prototyping
- Flexible Partnering
- Effective Support

Atgen

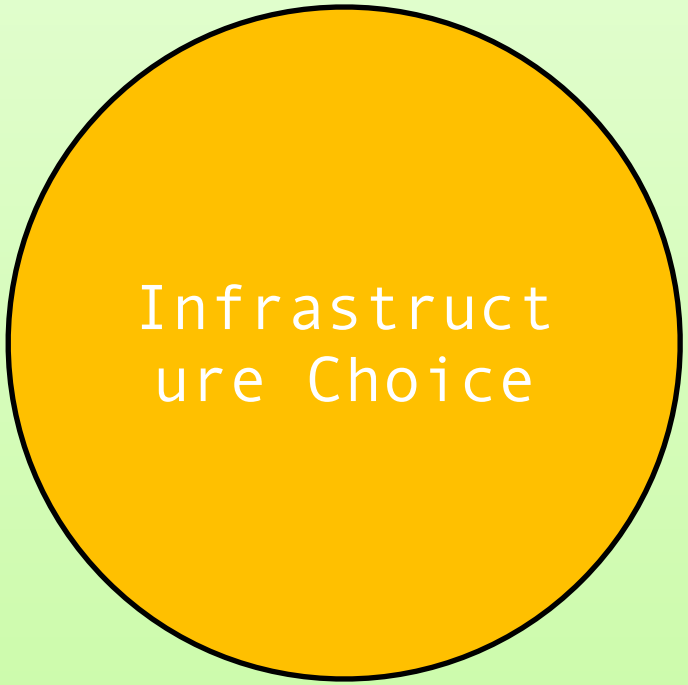
DevOps – Technical Innovation



- Polyglot Enablement
- DevOps Automation
- API Support
- Micro services Architecture
- Application Scaling and Elasticity
- PaaS

Atgen

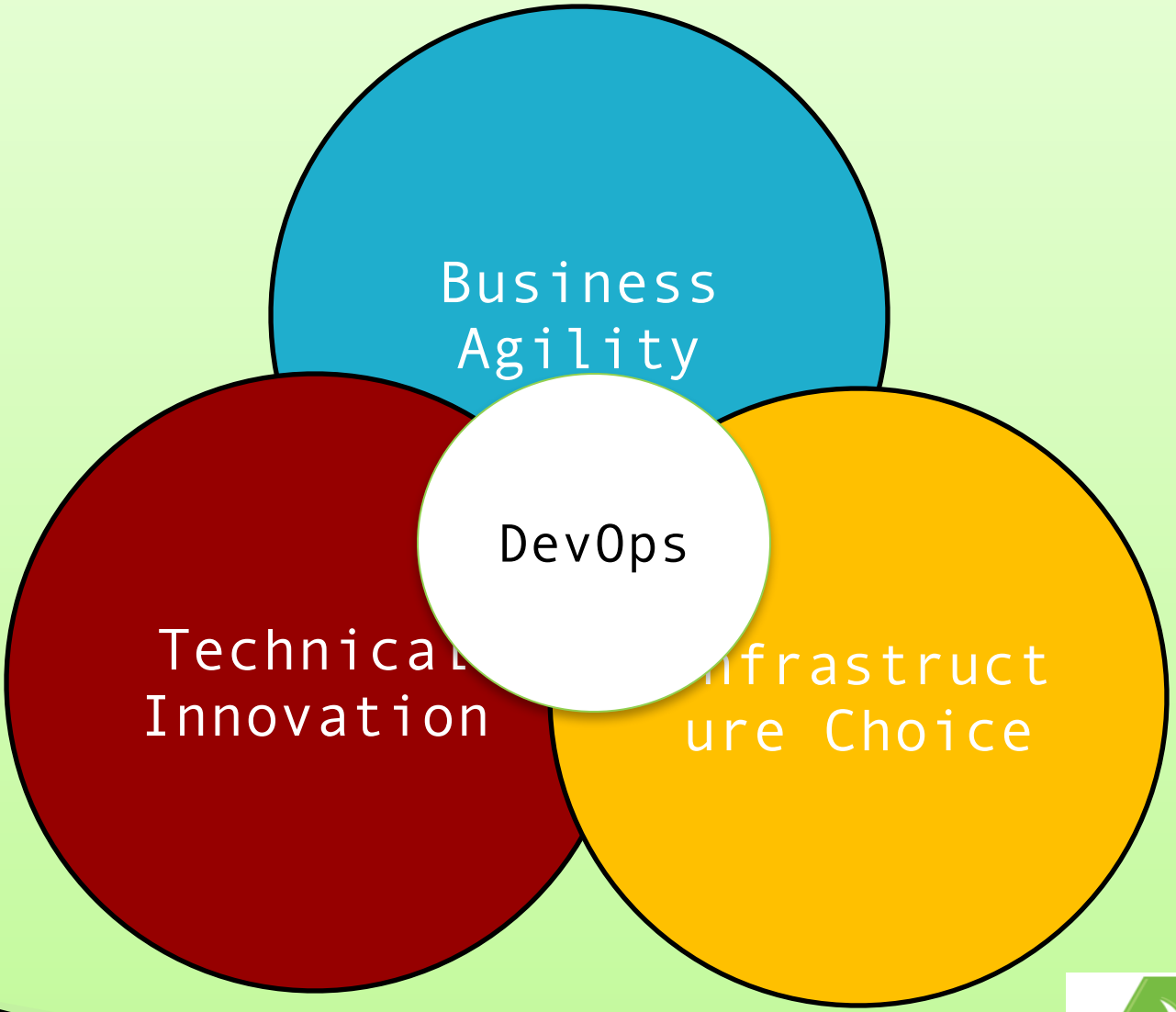
DevOps – Technical Innovation



- Docker Foundation
- Language and Stack Neutral
- Lightweight Hybrid Cloud with AWS, VMware, & OpenStack
- Common Application Design and Operations

Atgen

DevOps - Technical Innovation



Atgen

DevOps – Culture is Yoghurt

- Culture is an output, not an input
- Common organization with a nearby boss
- Shared metrics foundation of collaboration
- High-level sponsorship

Atgen

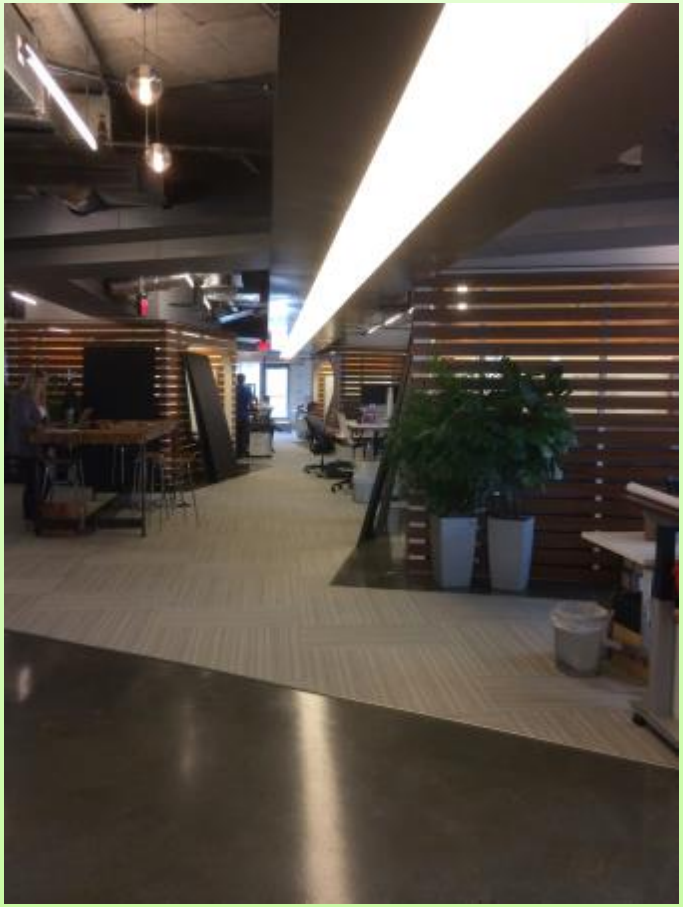
DevOps – Don't Build New Legacy

- Smart person + 3 years: "How's that system work?"
- Future-proof your application pipeline
- Seek commercial product

Atgen

DevOps – Innovation Lab

- Separate Organization
- Accelerate Adoption of New Technologies
- Exploration via New Skills and Employees
- Integration Process Defined



Atgen

DevOps – Benefits

IT Domain	Before DevOps	After DevOps
Application lifecycle	Months	Weeks
Polyglot support	Java only	10+ languages/ stacks
Integration of Innovation Lab with mainstream IT	Limited	Structured
Attractiveness to new developers	Minimal	Significant

Atgen

The Essence of DevOps

- 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

"Break down the wall between development and operations"

Atgen

Session: 3

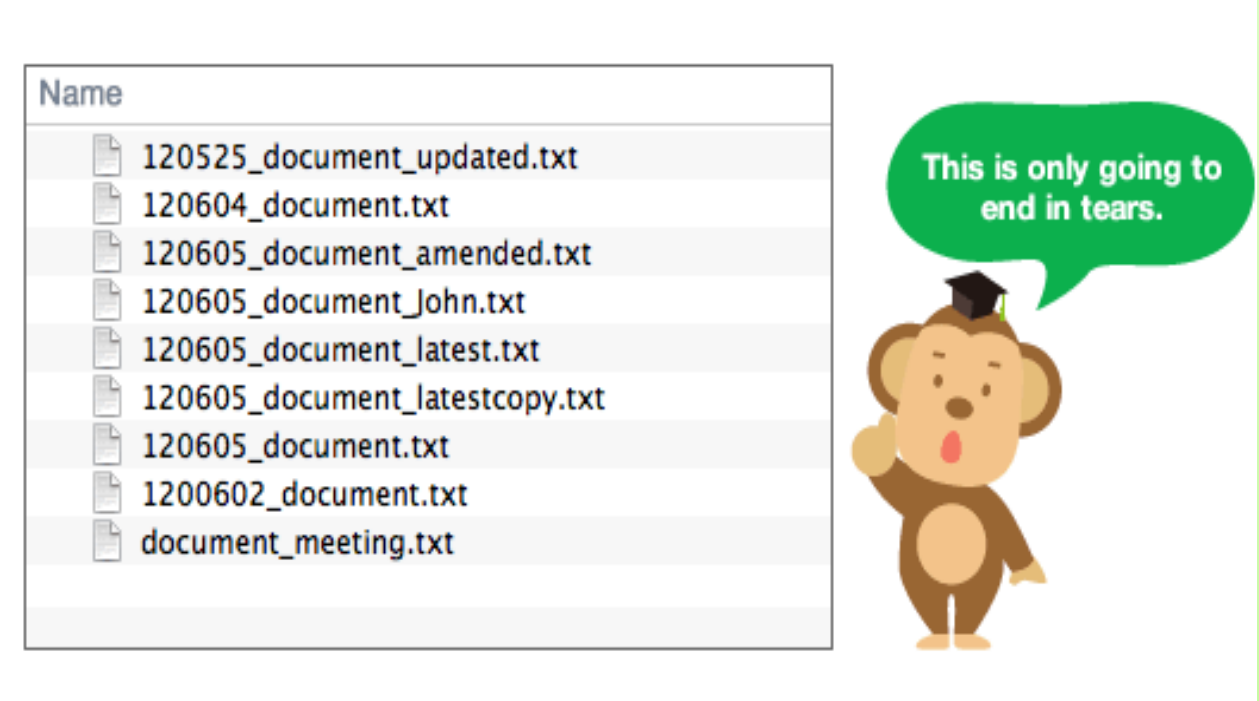
VCS & GIT

Introduction

- Version control systems are a category of software tools that help a team manage changes to source code over time.
- Version control software keeps track of every modification to the code in a special kind of database.
- Version control helps to keep tracking every individual change by each contributor and helping prevent concurrent work from conflicting
- It allows us to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

Why VCS ?

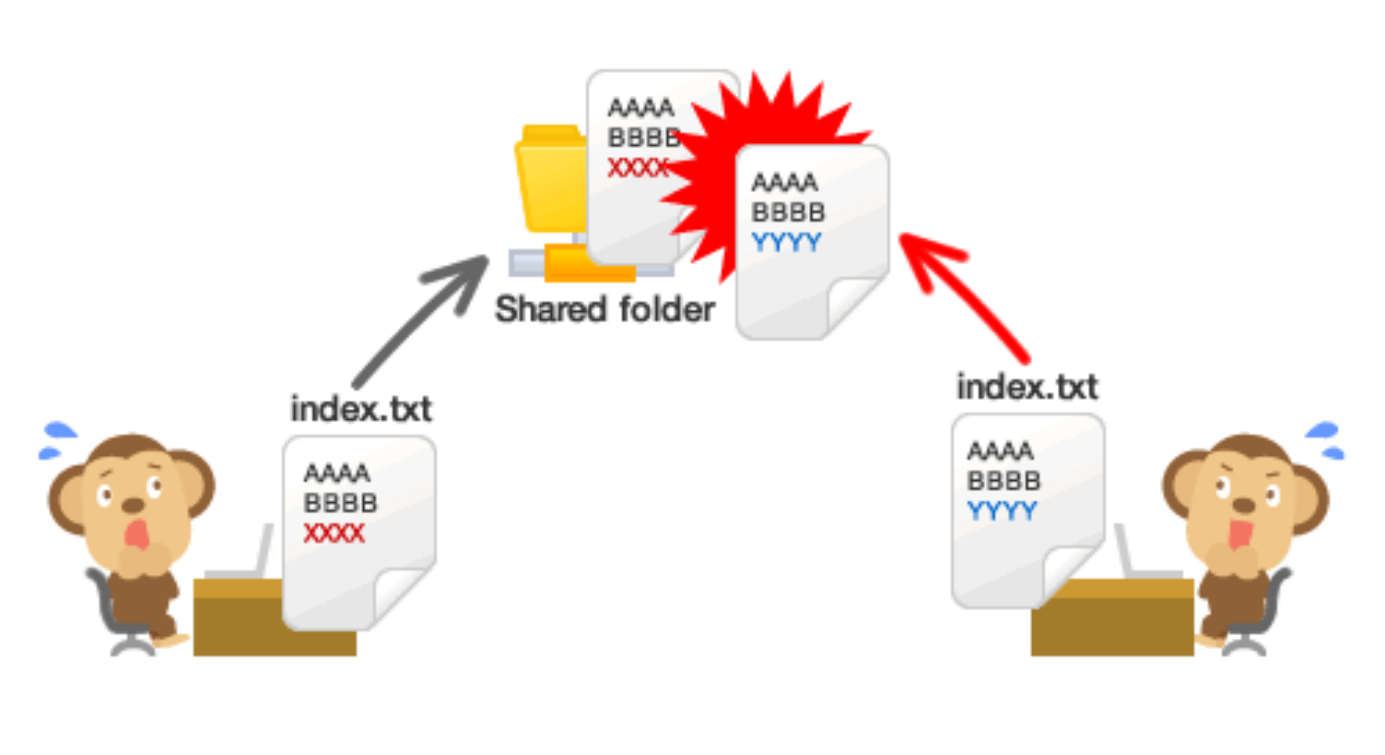
- Storing Versions (Properly)
- Restoring
- Backup



Atgen

Why VCS ?

- Merging changes



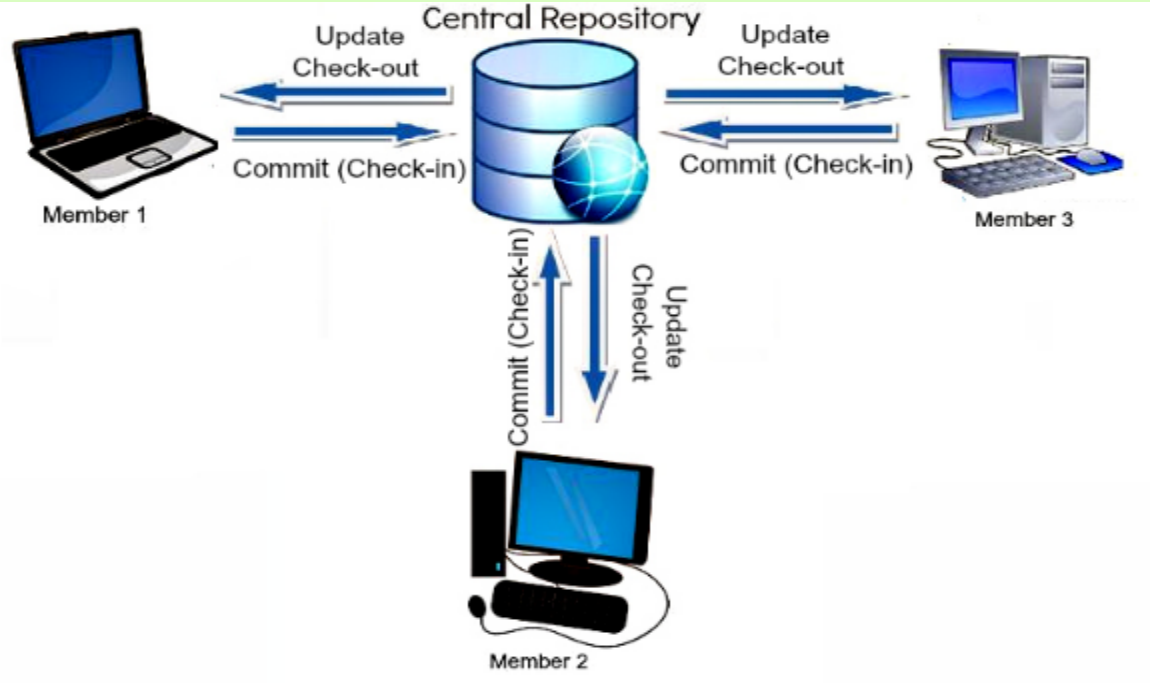
Atgen

Benefits

- A complete long-term change history of every file
- Traceability
- Manageability
- Restoring Easiness
- Collaborative Work

CVCS vs DVCS

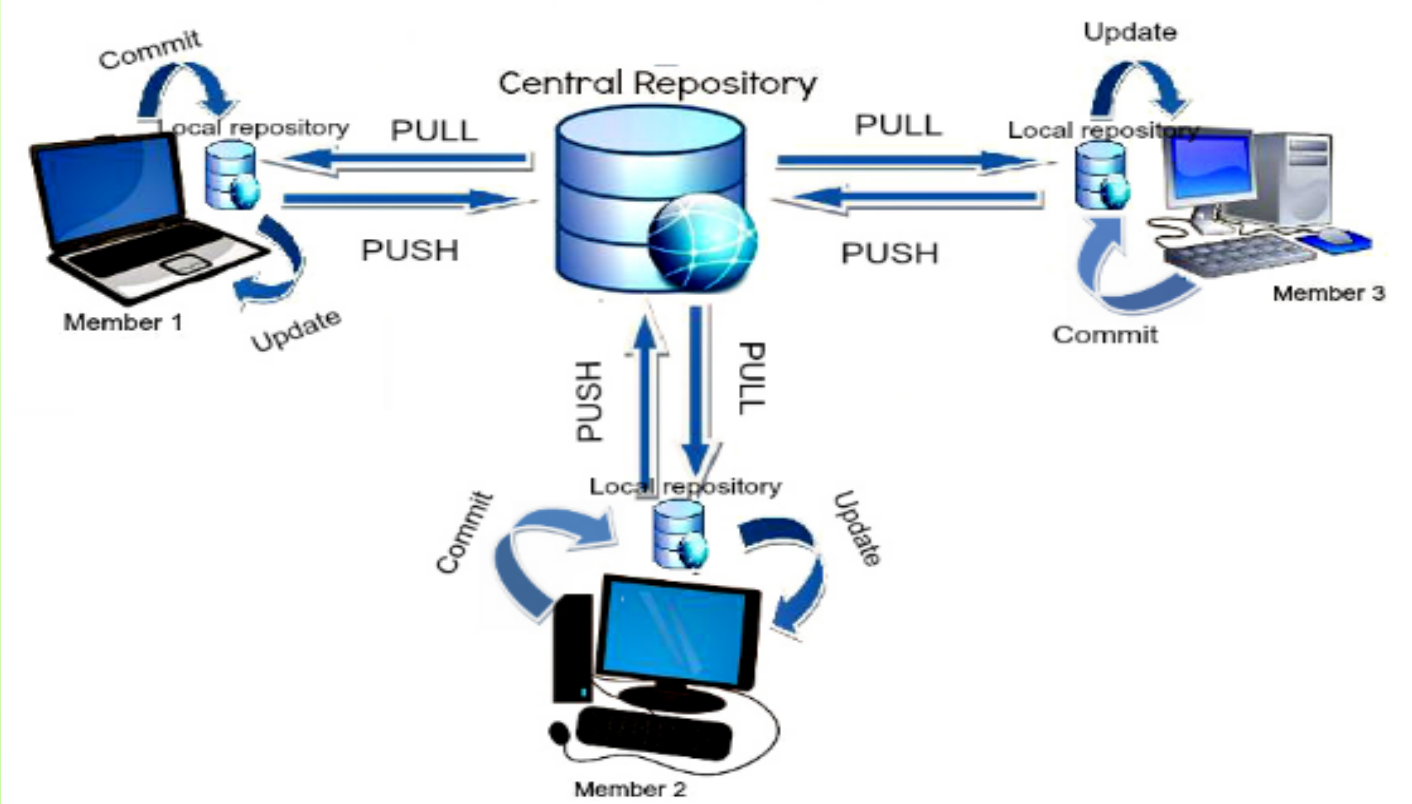
- CVCS(Centralized)
- Client-Server relationship. The repository is located at one place and provides access to many clients.



Atgen

CVCS vs DVCS

- DVCS(Distributed)

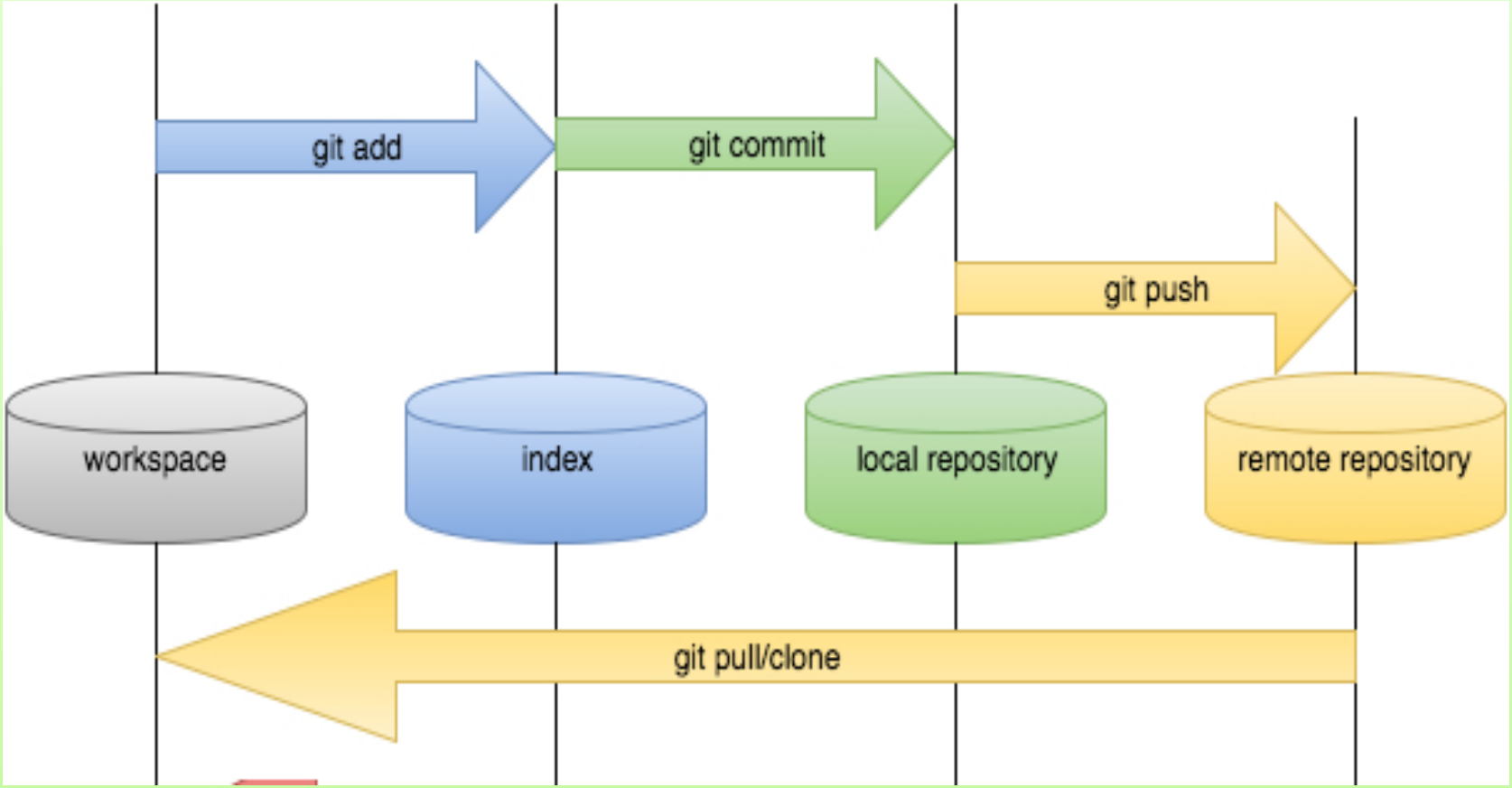


Atgen

What is GIT

- Distributed version controlling system
- Design for the challenges of large-scale
- Open-source project
- Support fast
- off-line work

GIT repository and workflow



Atgen

GIT – Classroom Environment

- Installation
  - If you're on Fedora/CentOS for example, you can use yum:

    ```
    sudo yum install git
    ```

  - If you're on a Debian-based distribution like Ubuntu, try apt-get:

    ```
    sudo apt-get install git-all
    ```

GIT – Classroom Environment

- Configuration
  - Identity

    ```
    git config --global user.name “Sagar Mehta“
    git config --global user.email sagar.mehta@atgensoft.com
    ```

  - Editor

    ```
    git config --global core.editor vim
    ```

  - Verify Config

    ```
    git config –list
    ```

GIT – Classroom Environment

- Creating Repository
  - Workspace

    ```
    git init
    ```

  - Bare(Remote)

    ```
    git --bare init
    ```

GIT – Classroom Environment

- Create & manage remote repository
  - Add a file

    ```
    git add filename
    ```

  - Check Status

    ```
    git status
    ```

  - Commit

    ```
    git commit –m “message” –a
    ```

  - Verify Logs

    ```
    git log
    ```

GIT – Classroom Environment

- Managing Remote Repository
  - Create remote repository

    ```
    git init --bare <foldername>
    ```

  - Add remote repository

    ```
    git remote add <name> <url>
    ```

  - Verify Remote Repository

    ```
    git remote -v
    ```

GIT – Classroom Environment

- Managing Remote Repository
  - Remove Remote Repository

    ```
    git remote rm <name>
    ```

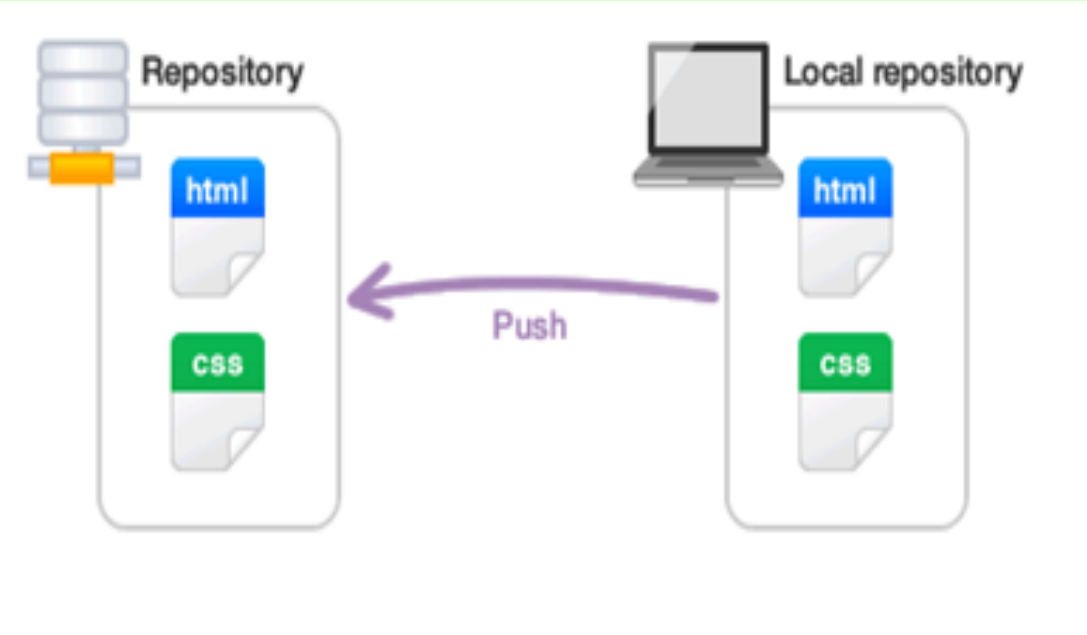
  - Rename Remote Repository

    ```
    git remote rename <old-name> <new-name>
    ```

GIT – Classroom Environment

- Push Data to Remote Repository
  - Push to Remote Repository

```
git push <remote> <branch>
```

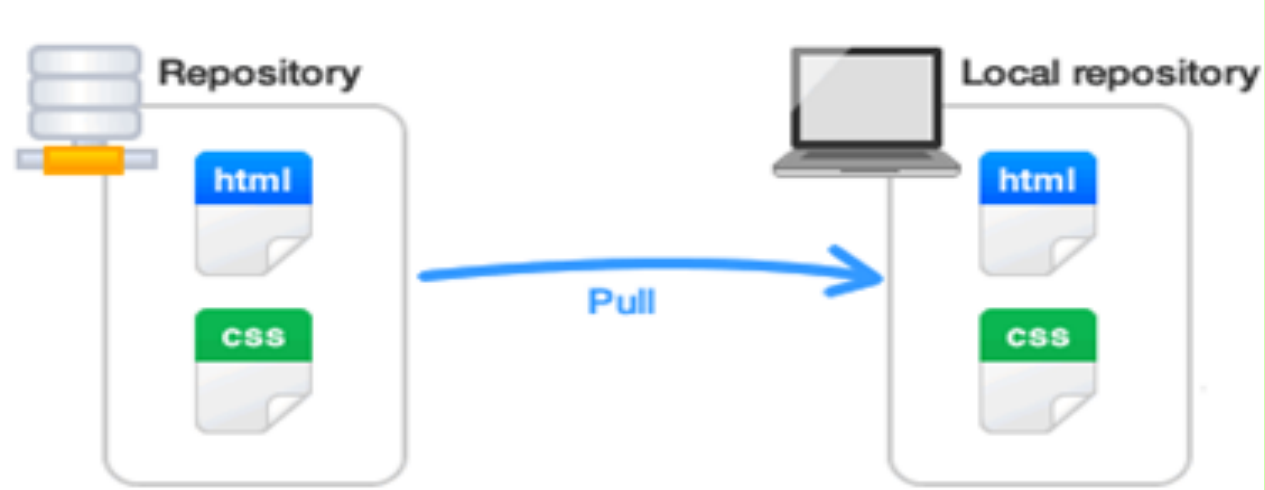


Atgen

GIT – Classroom Environment

- Pull
  - Pull from Remote Repository

```
git pull <remote> <branch>
```



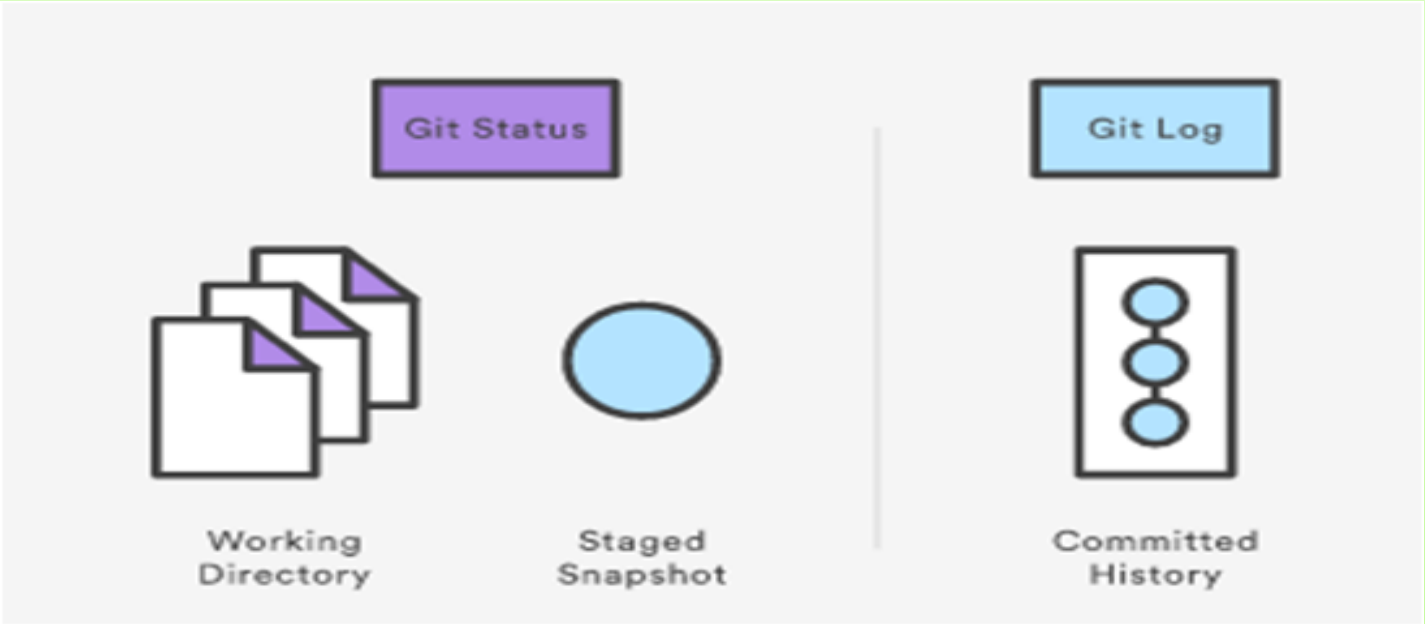
Atgen

GIT – Classroom Environment

- Status
  - Check Status

    git status
  - Verify Logs

    git log



Atgen

GITHub

- GitHub Enterprise is the on-premises version of GitHub.
- With flexible deployment options, centralized permissions, and hundreds of integrations, you and your team can enjoy the best parts of working with GitHub without compromising on the features your business needs.
- http://github.com/ is place for cloud Git Account.

Session: 4

DevOps - CM

Orchestrators at a Glance

	Puppet 3.0	Chef 11.4	Ansible 1.3	Salt 0.17
Pros	• Modules can be written in Ruby or a simpler, Puppet-specific subset of Ruby • Push commands let you trigger modifications immediately • Web UI handles reporting, inventorying, and real-time node management • Detailed and in-depth reporting on agent runs and node configurations	• Cookbooks and recipes can leverage the full power of Ruby • Centralized JSON-based "data bags" allow scripts to populate variables during runtime • Web UI lets you search and inventory nodes, view node activity, and assign Cookbooks, roles, and nodes	• Modules can be written in nearly any language • No agent required on managed clients • Web UI lets you configure users, teams, and inventories, and apply Playbooks to inventories • Extremely simple to set up and get running	• State files can be simple YAML configuration templates or complex Python or PyDSL scripts • Can communicate with clients through SSH or through a locally installed agent • Web UI offers views of running jobs, minion status, and event logs, and lets you execute commands on clients • Extremely scalable

Orchestrators at a Glance

	Puppet 3.0	Chef 11.4	Ansible 1.3	Salt 0.17
Cons	• Requires learning Puppet DSL or Ruby • Installation process lacking in error checking and error reporting	• Requires knowledge of Ruby programming • Currently lacks functional push commands • Documentation is sometimes vague	• Lacks support for Windows clients • Web UI doesn't tie into an existing Ansible deployment automatically; inventories must be imported	• Web UI is not as mature or complete as competitors • Lacks deep reporting capabilities
Pricing	Free open source version; Puppet Enterprise costs $100 per machine per year	Free open source version; Enterprise Chef free for 5 machines, $120 per month for 20 machines, $300 per month for 50 machines, $600 per month for 100 machines, and so on	Free open source version; AWX free for 10 machines, then $100 or $250 per machine per year depending on support	Free open source version; SaltStack Enterprise costs $150 per node per year, with volume discounts and site licenses available

Ansible - History

- Michael DeHaan, the author of the provisioning server application Cobbler and co-author of the Func framework for remote administration, developed the platform on 20th Feb' 2012.
- It is included as part of the Fedora distribution of Linux, owned by Red Hat Inc., and is also available for Red Hat Enterprise Linux, CentOS, and Scientific Linux via Extra Packages for Enterprise Linux (EPEL) as well as for other operating systems.
- Ansible, Inc. (originally AnsibleWorks, Inc.) was the company set up to commercially support and sponsor Ansible.
- It was acquired by Red Hat in October 2015.

Introduction

- Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.
- Being designed for multi-tier deployments since day one, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time.
- It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

Ansible- Efficient Architecture

- Ansible works by connecting to your nodes and pushing out small programs, called "Ansible Modules" to them.
- These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished.
- Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.

Ansible Tower

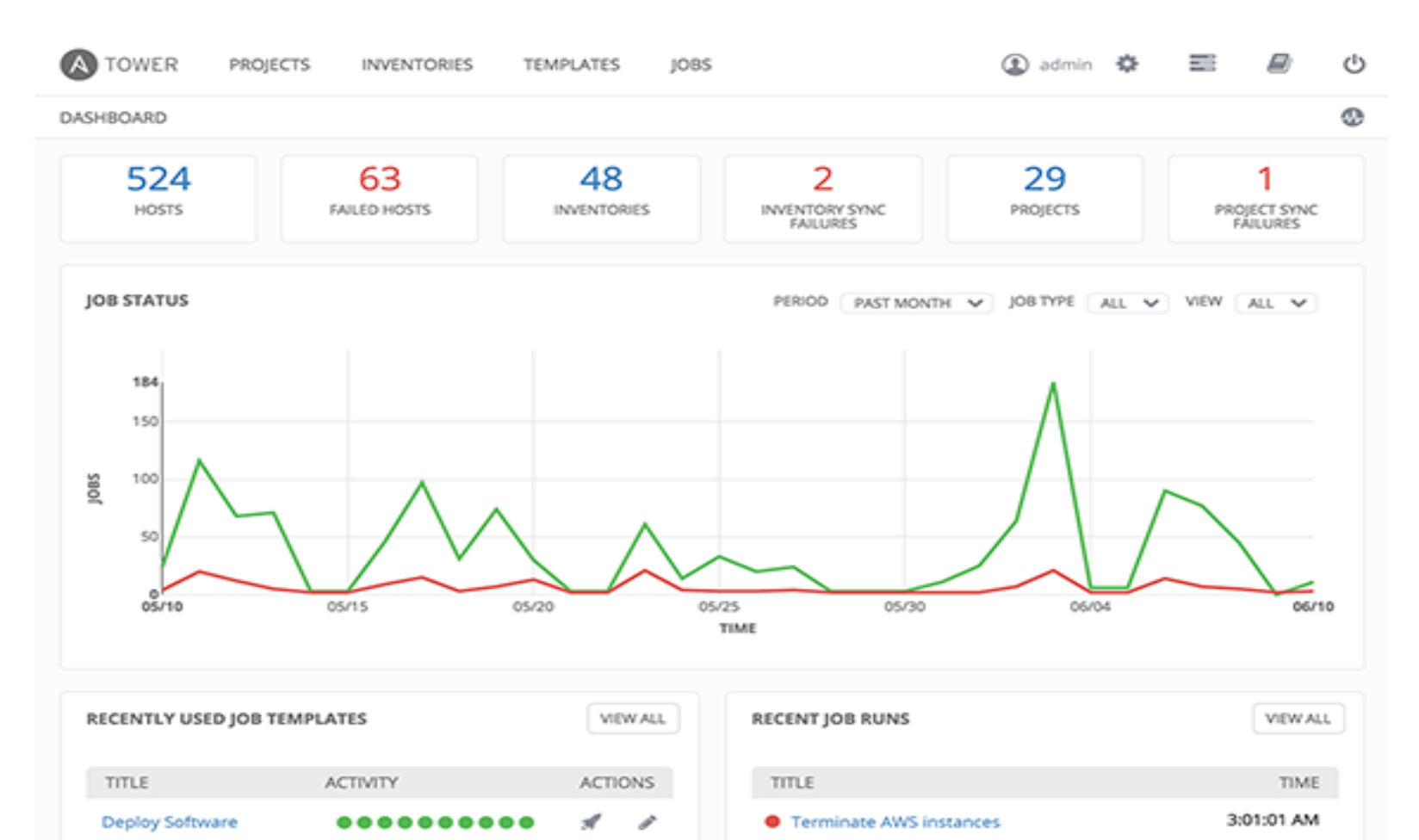
- Ansible Tower by Red Hat helps you scale IT automation, manage complex deployments and speed productivity.
- Centralize and control your IT infrastructure with a visual dashboard, role-based access control, job scheduling and graphical inventory management.
- Tower now features a streamlined interface, integrated notifications and enhanced permissions.
- Tower's REST API and CLI make it easy to embed Tower into existing tools and processes

Ansible Tower

- Ansible Dashboard
  - The Tower dashboard provides a heads-up NOC-style display for everything going on in your Ansible environment. You'll see your host and inventory status, all the recent job activity and a snapshot of recent job runs.

- Real Time Job Status Updates
  - Within Tower, Playbook runs stream by in real time. As Ansible automates across your infrastructure, you'll see plays and tasks complete, broken down by each machine, and each success or failure, complete with output. Easily see the status of your automation, and what's next in the queue.

Ansible Tower



Atgen

Ansible – Working Model

Inventory File

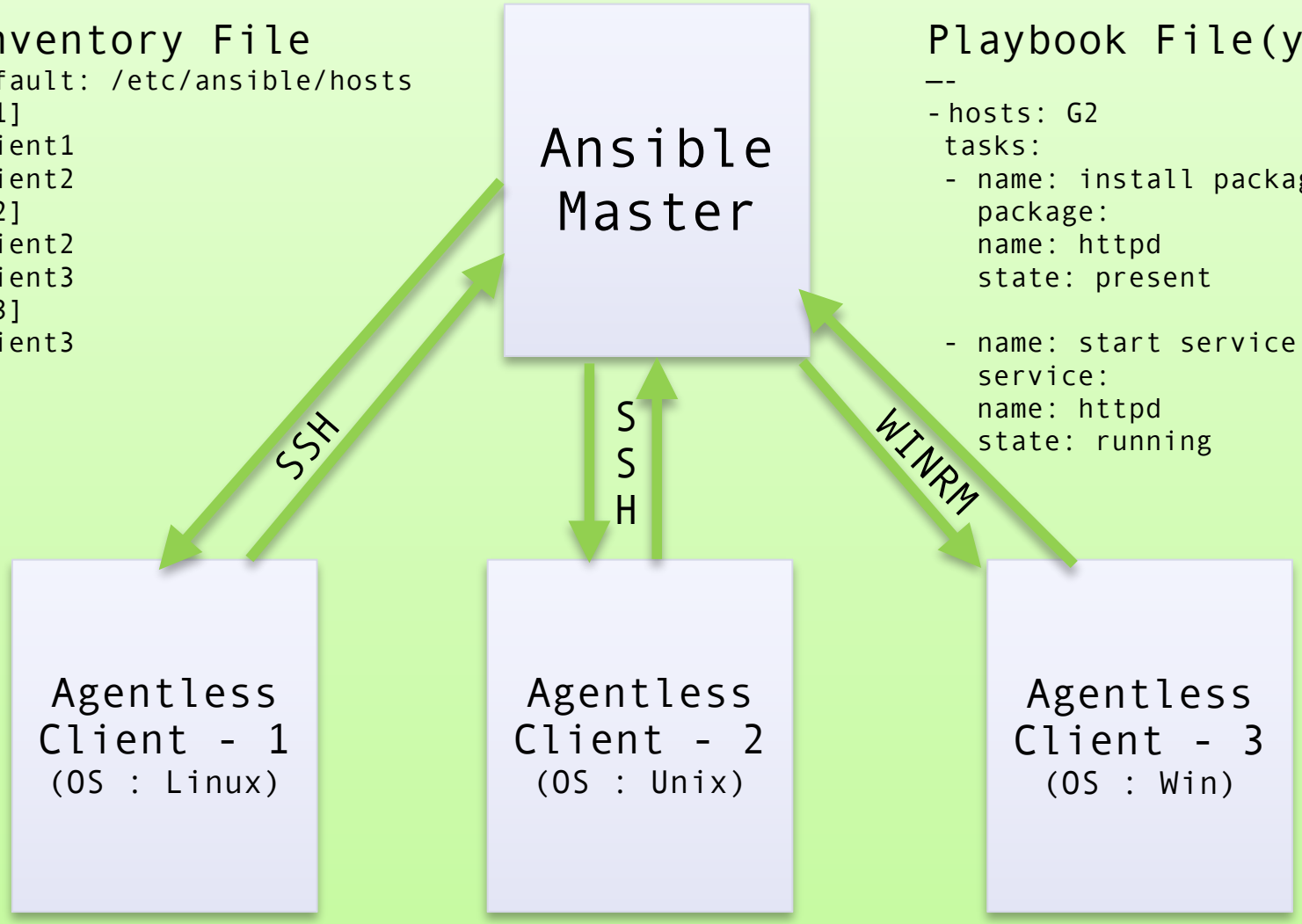
Default: /etc/ansible/hosts

```
[G1]
client1
client2
[G2]
client2
client3
[G3]
client3
```

Playbook File(yaml)

```
---
- hosts: G2
  tasks:
  - name: install package
    package:
    name: httpd
    state: present

  - name: start service
    service:
    name: httpd
    state: running
```



Atgen

Ansible – Inventory

- Ansible represents what machines it manages using a very simple INI file that puts all of your managed machines in groups of your own choosing.
- To add new machines, there is no additional SSL signing server involved, so there's never any hassle deciding why a particular machine didn’t get linked up due to obscure NTP or DNS issues.
- Here's what a plain text inventory file like:

```
[webservers]
www1.example.com
www2.example.com

[dbservers]
db0.example.com
db1.example.com
```

Atgen

Ansible – Playbooks

- Playbooks can finely orchestrate multiple slices of your infrastructure topology, with very detailed control over how many machines to tackle at a time.
- Ansible's approach to orchestration is one of finely-tuned simplicity, as we believe your automation code should make perfect sense to you years down the road and there should be very little to remember about special syntax or features.

An Example APP_CONFIG.YML

```
---
- yum: name={{item}} state=installed with_items:
   - app_server
   - acme_software

- service: name=app_server state=running
enabled=yes
```

Playbooks

- Playbooks are Ansible's configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.
- If Ansible modules are the tools in your workshop, playbooks are your design plans.
- At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.
- Playbooks are designed to be human-readable and are developed in a basic text language.

Playbooks

- Playbooks are a completely different way to use ansible than in adhoc task execution mode, and are particularly powerful.
- Simply put, playbooks are the basis for a really simple configuration management and multi-machine deployment system, unlike any that already exist, and one that is very well suited to deploying complex applications.
- Playbooks can declare configurations, but they can also orchestrate steps of any manual ordered process, even as different steps must bounce back and forth between sets of machines in particular orders. They can launch tasks synchronously or asynchronously.
- While you might run the main ”/usr/bin/ ansible” program for ad-hoc tasks, playbooks are more likely to be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.

Playbooks Language Example

- Playbooks are expressed in YAML format and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process.
- Each playbook is composed of one or more ‘plays’ in a list.
- The goal of a play is to map a group of hosts to some well defined roles, represented by things ansible calls tasks. At a basic level, a task is nothing more than a call to an ansible module.
- By composing a playbook of multiple ‘plays’, it is possible to orchestrate multi-machine deployments, running certain steps on all machines in the webservers group, then certain steps on the database server group, then more commands back on the webservers group, etc.
- You can have quite a lot of “plays” that affect your systems to do different things. It’s not as if you were just defining one particular state or model, and you can run different plays at different times.
- For starters, here’s a playbook that contains just one play:

Playbooks Language Example

- Here's a playbook that contains just one play:

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is latest version
    package: name=httpd state=latest
  - name: write the apache config file
    copy: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: ensure apache is running
    service: name=httpd state=started enabled=yes
```

Playbooks with multiple plays

- Here's a playbook that contains just two plays:

```
---
- hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
yum: name=httpd state=latest
  - name: write the apache config file
    copy: src=/srv/httpd.j2 dest=/etc/httpd.conf

- hosts: databases
  remote_user: root

  tasks:
  - name: ensure postgresql is at the latest version
    yum: name=postgresql state=latest
  - name: ensure that postgresql is started
    service: name=postgresql state=started
```

Playbook –Implement a playbook

- A playbook can be implemented on hosts using below command:

```
ansible-playbook --syntax-check playbook.yml
ansible-playbook --check playbook.yml
ansible-playbook playbook.yml
```

Modules –Help Page

- All modules documentation can be seen using below command:

```
ansible-doc copy
```

Playbook –Create User on Host

- A task to create user & group on Host by writing a playbook.

Copy Files

- The copy module copies a file on the local box to remote locations.
- Use the fetch module to copy files from remote locations to the local box.

```
# Example from Ansible Playbooks
- copy: src=/srv/myfiles/foo.conf dest=/etc/foo.conf owner=foo group=foo
  mode=0644

# Copy a new "ntp.conf file into place, backing up the original if it differs from the copied version
- copy: src=/mine/ntp.conf dest=/etc/ntp.conf owner=root group=root mode=644
  backup=yes
```

Ignoring Failed Commands

- Generally playbooks will stop executing any more steps on a host that has a failure. Sometimes, though, you want to continue on.
- To do so, write a task that looks like this:

```
- name: this will not be counted as a failure
  command: /bin/false
  ignore_errors: yes
```

Playbook – Setup Tomcat

- Write a playbook to install tomcat with “sample.war” to test it.

Session: 5

DevOps – Continuous Testing

“By tightly connecting testing with development and operations, and automating the design, development, quality assurance and deployment of new applications and systems, IT organizations can more effectively collaborate and deliver on the dual mandate of increasing operational agility and speeding software’s time to market.”

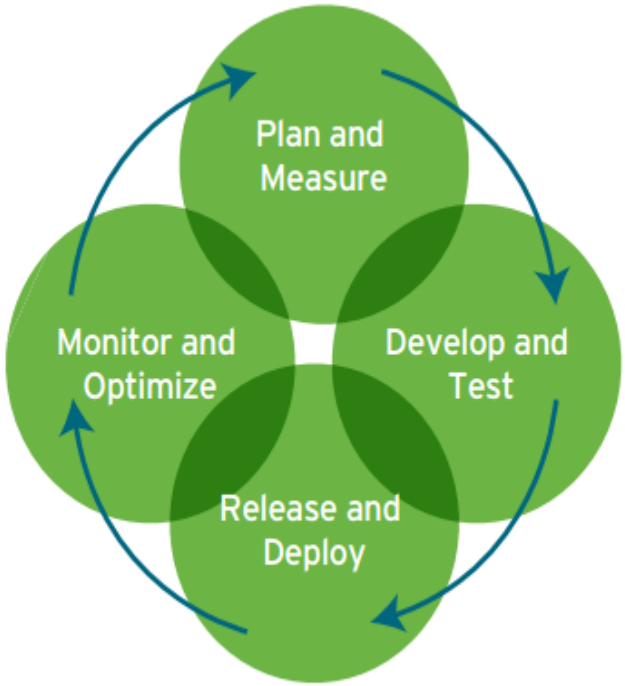
DevOps – At a Glance

Collaborative Development:
Increased collaboration between teams

Continuous Integration & Continuous Testing:
Integration of software testing with deployment and operations

Continuous Release and Deployment:
Increased delivery speed and frequency

Continuous Monitoring:
Improved quality by monitoring production performance



Atgen

Agile vs DevOps

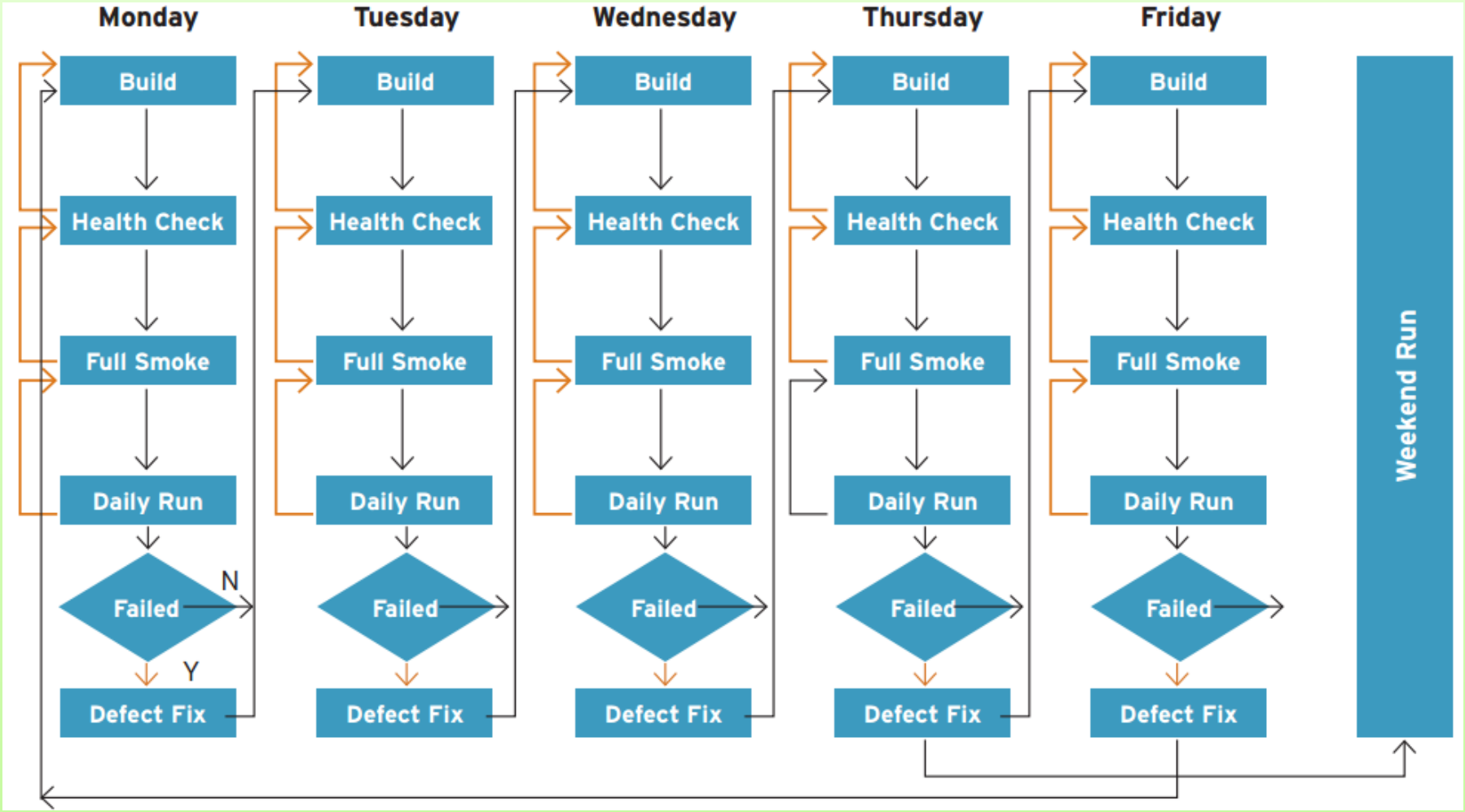
Testing in Agile	Testing in DevOps
Test as early and as often as possible	Test continuously
Automate testing as much as possible	Automate almost everything
Continuous integration and testing is a step forward	Continuous integration and testing is mandatory
Potentially shippable code at the end of a sprint	Potentially shippable code following every integration

Principles –Test Automation

- A dynamic regression scope for each build based on the type of code change; automatically generated release notes will feed into the regression job.
- An intelligent test automation framework capable of turning test scenarios on and off based on application changes.
- Test cases traceable to the code files.
- A centralized repository of code-to-test case traceability.
- A risk-based approach with near zero risk.
- Integration of Functional & Non-functional testing with CT.

Atgen

Intelligent Testing Schedule



Atgen

Continuous Testing - Virtues

- Improved Availability
- Faster Deployment
- Accelerate Error Detection

Continuous Testing - In Practice

- Tools, Scripting and Overall Test Creation
- Provisioning & Resources
- Automation
- On-Demand Testing and Re-Running Failed Use Cases
- Version Control Friendly Incremental Testing
- Automatic ‘Failure’ Alerts
- Reporting

Atgen

Session: 6

DevOps - Containerisation

History of Containers

- Containers have been around for a very long time indeed.
  - IBM VM/370 (1972)
  - FreeBSD jails (1999)
  - Linux VServers (2001)
  - Solaris Containers (2004)

Origins of Docker Project

- ‘dotCloud’ was operating a PaaS, using a custom container engine.
- This engine was based on ‘OpenVZ’ (and later, LXC) and AUFS.
- It started (circa 2008) as a single Python script.
- By 2012, the engine had multiple (~10) Python components. (and ~100 other micro-services!)
- End of 2012, ‘dotCloud’ refractors this container engine.
- The codename for this project is "Docker."

First Public Release

- March 2013, PyCon, Santa Clara:

  "Docker" is shown to a public audience for the first time.
- It is released with an open source license.
- Very positive reactions and feedback!
- The ‘dotCloud’ team progressively shifts to Docker development.
- The same year, ‘dotCloud’ changes name to Docker.
- In 2014, the PaaS activity is sold.

First Users of Docker(2013-14)

- PAAS builders (Flynn, Dokku, Tsuru, Deis...)
- PAAS users (those big enough to justify building their own)
- CI platforms
- developers, developers, developers, developers

Becomes industry standard(15-16)

- Docker reaches the symbolic 1.0 milestone.
- Existing systems like Mesos and Cloud Foundry add Docker support.
- Standards like OCI, CNCF appear.
- Other container engines are developed.

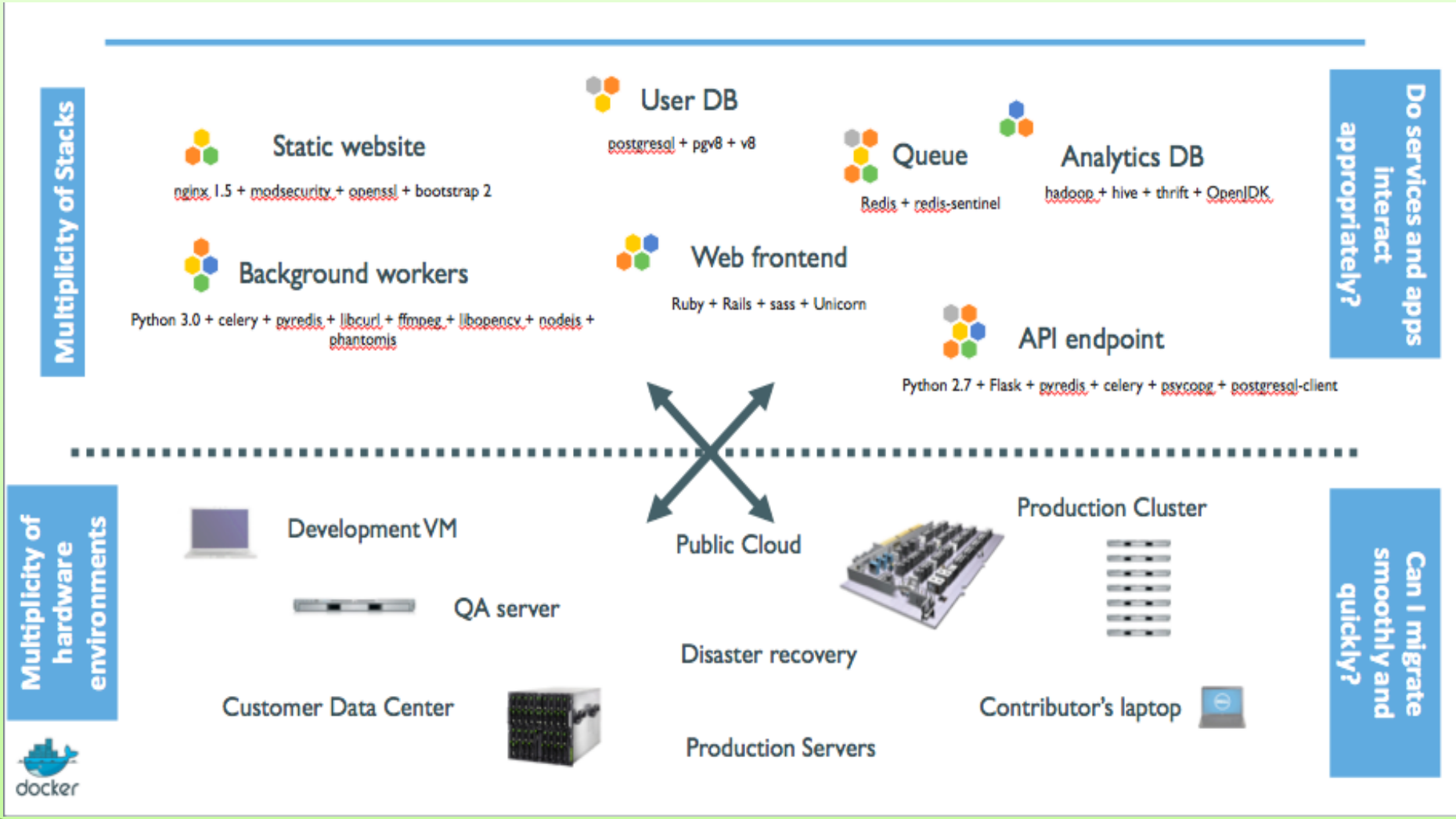
Docker becomes a platform

- The initial container engine is now known as "Docker Engine“.
- Other tools are added:
  - Docker Compose (formerly "Fig")
  - Docker Machine
  - Docker Swarm
  - Kitematic
  - Docker Cloud (formerly "Tutum")
  - Docker Datacenter
- Docker Inc. launches commercial offers.

About Docker Inc.

- Docker Inc. used to be ‘dotCloud’ Inc.
- ‘dotCloud’ Inc. used to be a French company.
- Docker Inc. is the primary sponsor and contributor to the Docker Project:
  - Hires maintainers and contributors.
  - Provides infrastructure for the project.
  - Runs the Docker Hub.
- HQ in San Francisco.
- Backed by more than 100M in venture capital.

Deployment Problem



Atgen

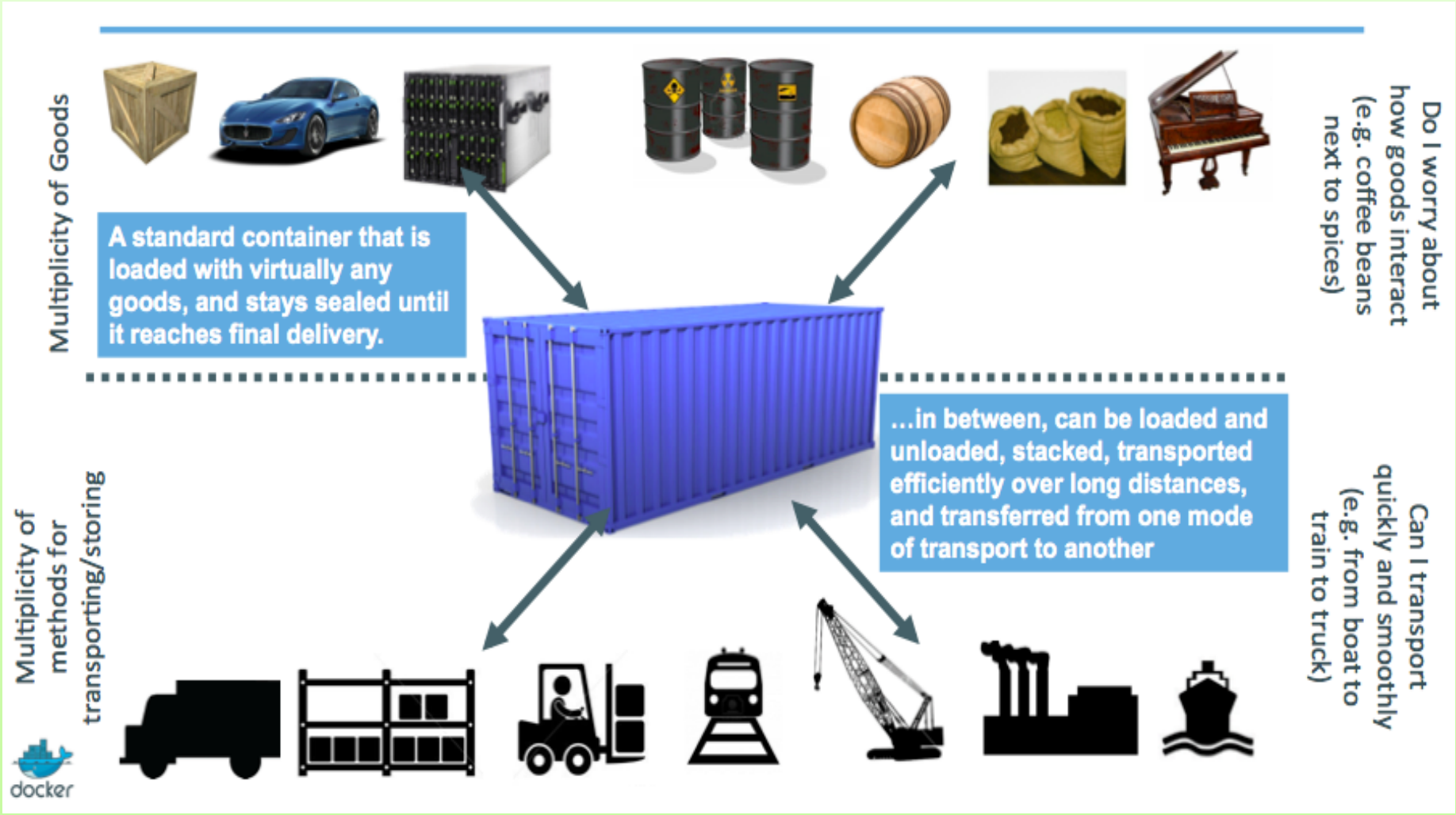
Matrix Checks

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

docker

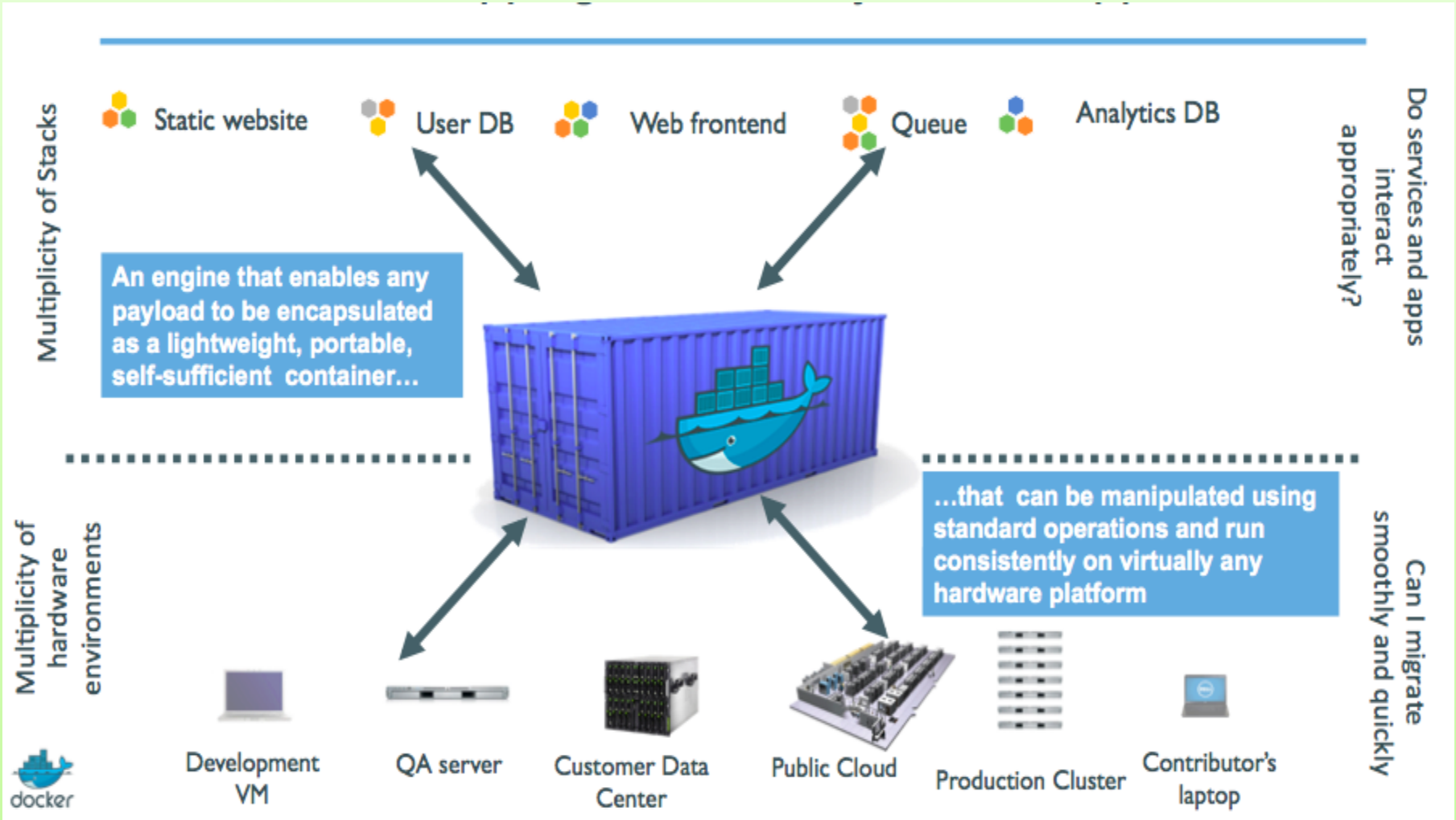
Atgen

Intermodal Shipping Containers



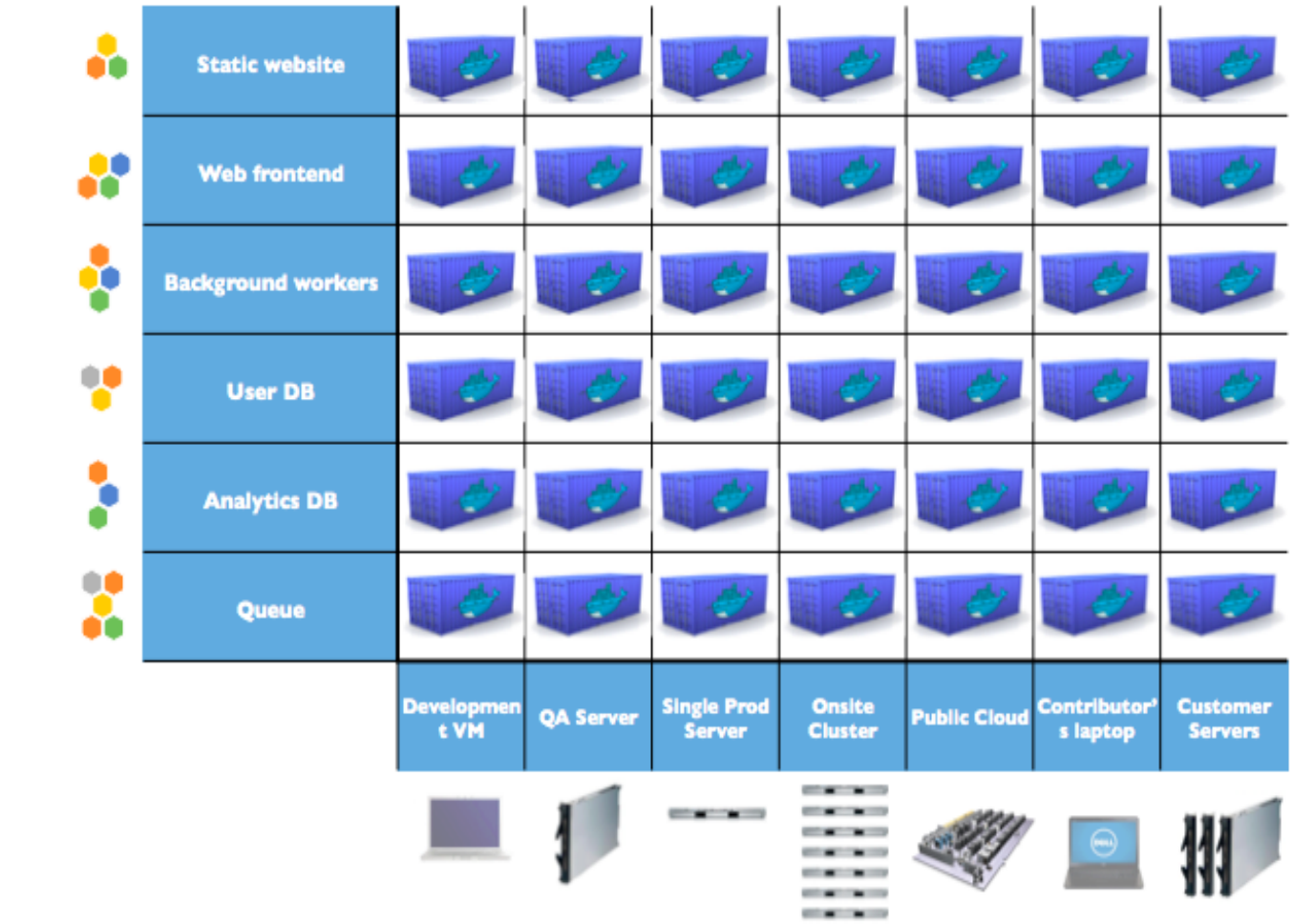
Atgen

Shipping Container for Applications



Atgen

Eliminate the Matrix



docker

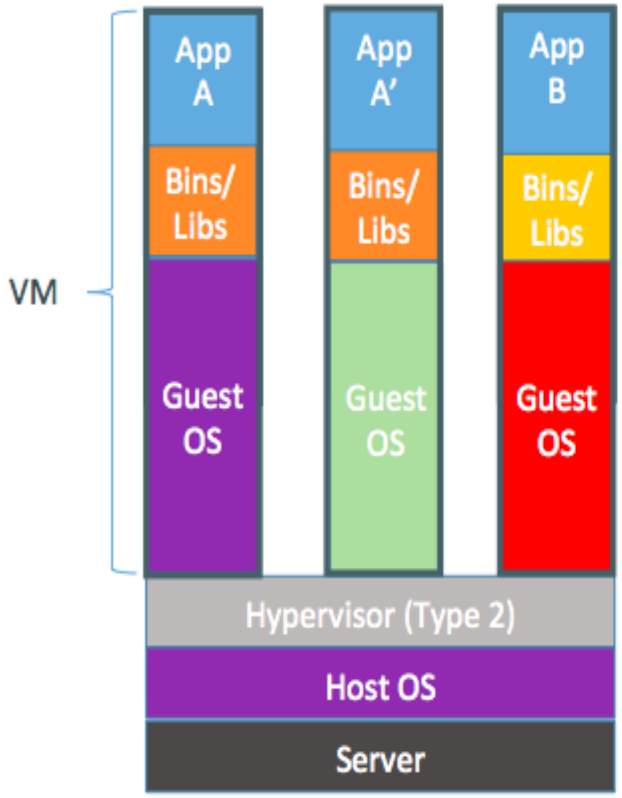
Atgen

Results

- Dev-to-prod reduced time.
- Continuous integration job time reduced by more than 60%.
- Better application management.

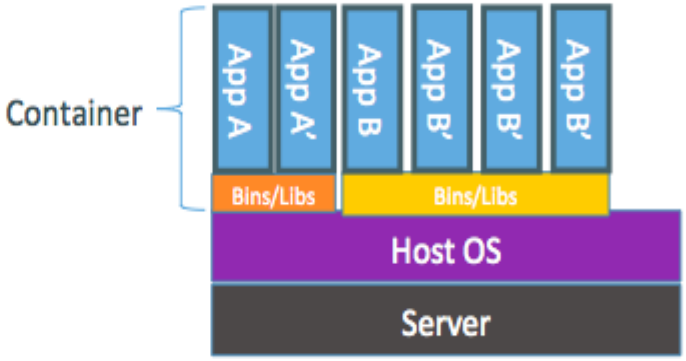
Atgen

Containers = cheaper than VMs



Containers are isolated,
but share OS kernel and, where appropriate, bins/libraries

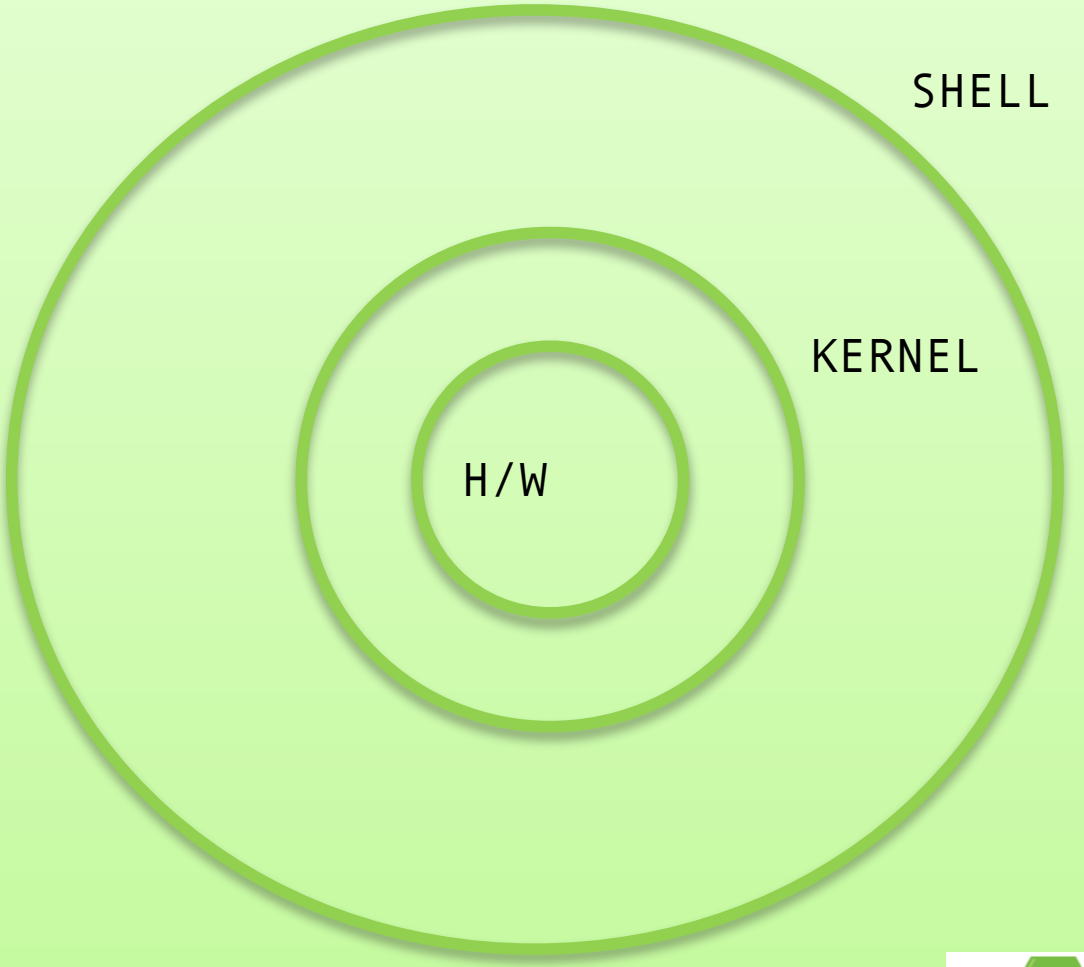
...result is significantly faster deployment, much less overhead, easier migration, faster restart



docker

Atgen

Containers = cheaper than VMs



Atgen

Docker Overview

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- With Docker, you can manage your infrastructure in the same ways you manage your applications.
- By using Docker's methodologies for shipping, testing, and deploying, you can reduce time of customer delivery.

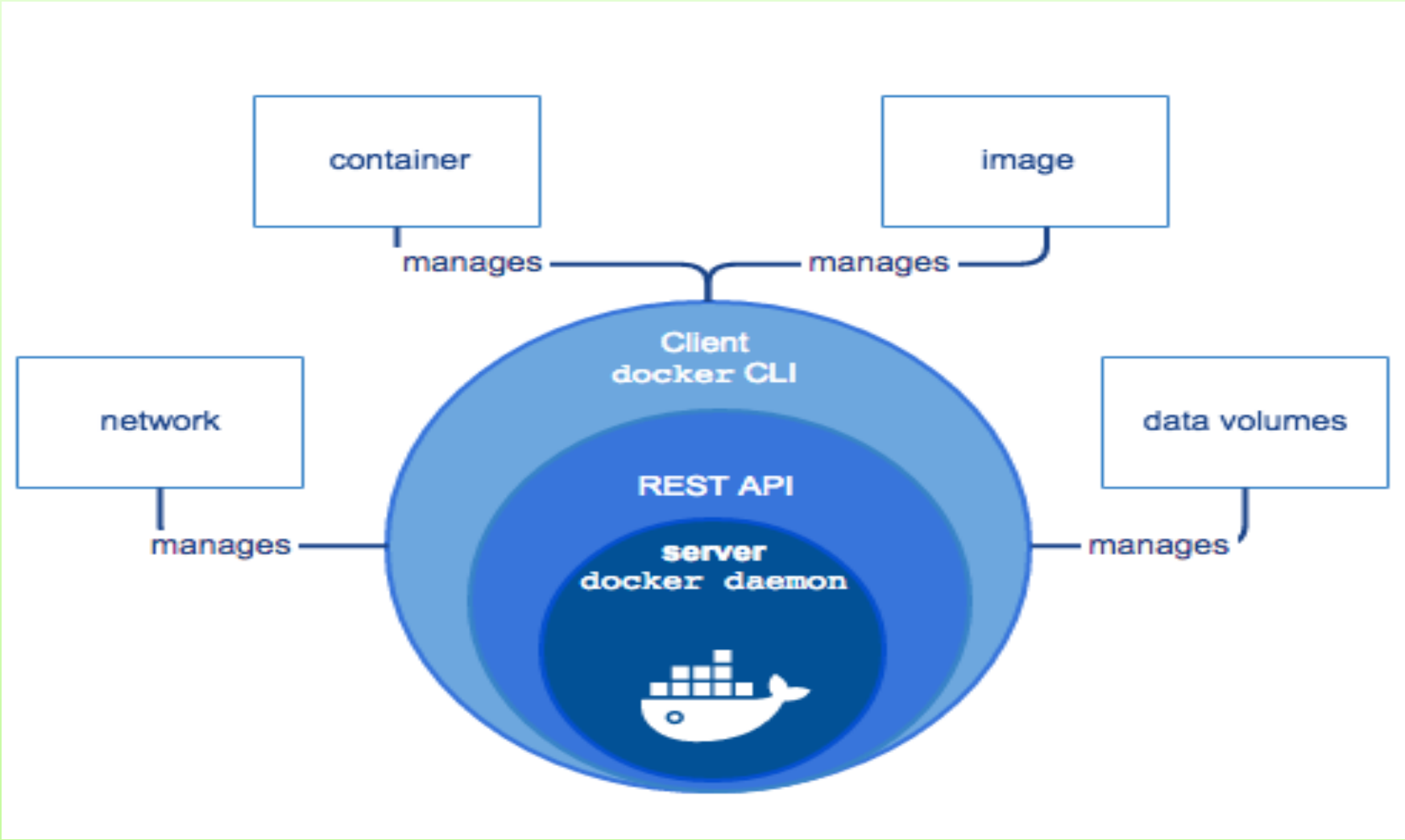
Docker Platform

- Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host.
- Because of the lightweight nature of containers, which run without the extra load of a hypervisor, you can run more containers on a given hardware combination than if you were using virtual machines.

Docker Platform

- Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Encapsulate your applications (and supporting components) into Docker containers
  - Distribute and ship those containers to your teams for further development and testing
  - Deploy those applications to your production environment, whether it is in a local data center or the Cloud

Docker Engine

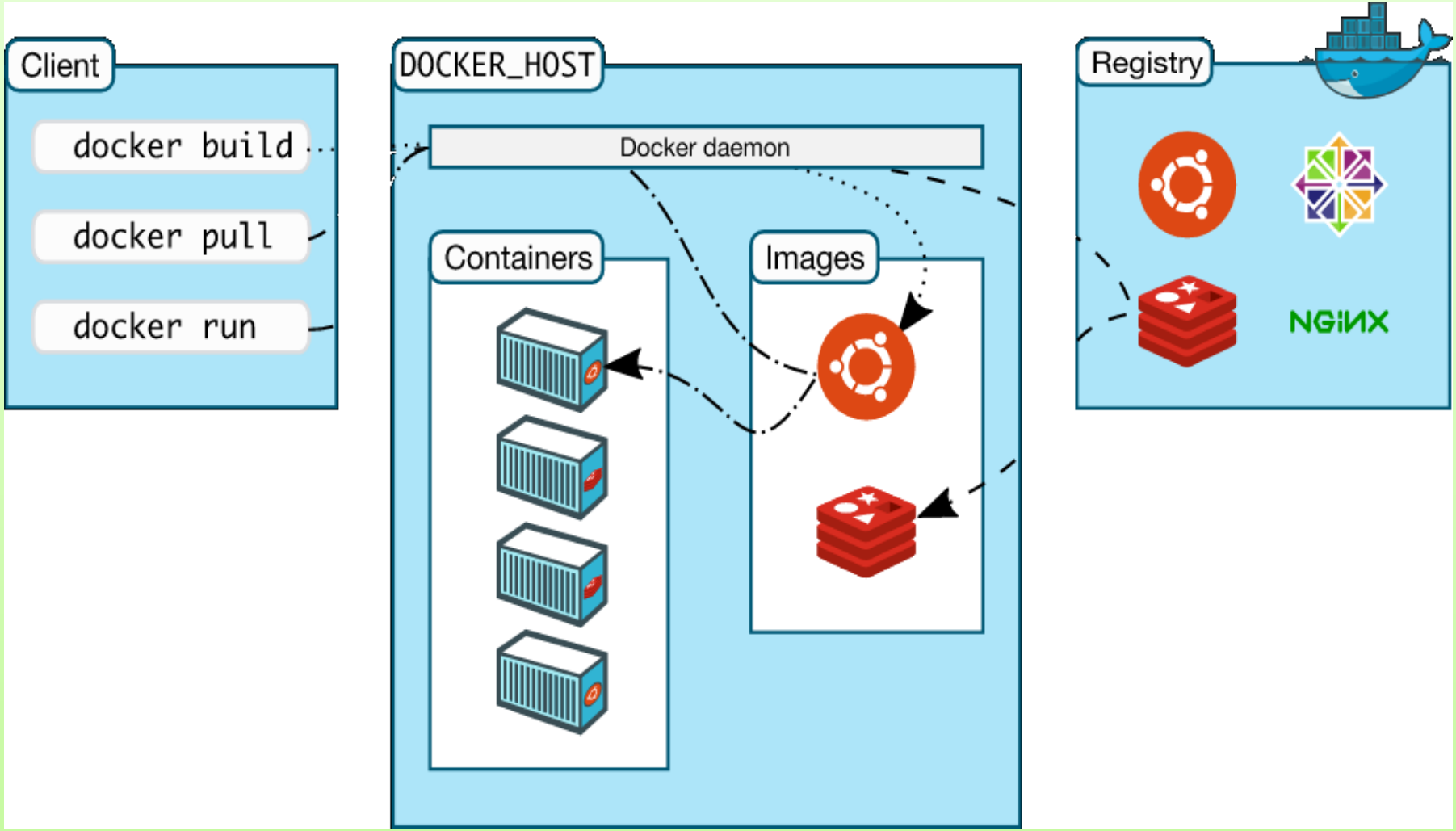


Atgen

Docker Engine

- Docker Engine is a client-server application with these major components:
  - A server which is a type of long-running program called a daemon process.
  - A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
  - A command line interface (CLI) client.
- The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands.
- The daemon creates and manages Docker objects, such as images, containers, networks, and data volumes.

Docker Architecture



Atgen

Docker Architecture

- Docker uses a client-server architecture.
- The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.
- The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Docker Architecture

- The Docker Daemon
  - The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.
- The Docker Client
  - The Docker client, in the form of the docker binary, is the primary user interface to Docker.
  - It accepts commands and configuration flags from the user and communicates with a Docker daemon.

Docker Images

- A Docker image is a read-only template with instructions for creating a Docker container.
- For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others.
- A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.
- Docker images are the build component of Docker.

Docker Containers

- A Docker container is a running instance of a Docker image.
- You can run, start, stop, move, or delete a container using Docker API or CLI commands.
- When you run a container, you can provide configuration metadata such as networking information or environment variables.
- Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases.
- Docker containers are the run component of Docker.

Docker Registries

- A docker registry is a library of images.
- A registry can be public or private, and can be on the same server as the Docker daemon or Docker client, or on a totally separate server.
- Docker registries are the distribution component of Docker.
- “Docker Hub” is known as global registry.

Docker Services

- A Docker service allows a swarm of Docker nodes to work together, running a defined number of instances of a replica task, which is itself a Docker image.
- You can specify the number of concurrent replica tasks to run, and the swarm manager ensures that the load is spread evenly across the worker nodes.
- To the consumer, the Docker service appears to be a single application.
- Docker services are the scalability component of Docker.

Docker Features

- Lightweight
  - Containers running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM. Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient.
- Open
  - Docker containers are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure.
- Secure
  - Containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application.

How Container works

- A container uses the host machine's Linux kernel, and consists of any extra files you add when the image is created, along with metadata associated with the container at creation or when the container is started.
- Each container is built from an image.
- The image defines the container's contents, which process to run when the container is launched, and a variety of other configuration details.
- The Docker image is read-only. When Docker runs a container from an image, it adds a read-write layer on top of the image (using a UnionFS) in which your application runs.

How Container works

- When you use the ”docker run” CLI command, the Docker Engine client instructs the Docker daemon to run a container.
- This example tells the Docker daemon to run a container using the ubuntu Docker image, to remain in the foreground in interactive mode (-i), and to run the /bin/bash command.

```
docker run -i -t ubuntu /bin/bash
```

How Container works

- When you run this command, Docker Engine does the following:
  - Pulls the ubuntu image
  - Creates a new container
  - Allocates a filesystem and mounts a read-write layer
  - Allocates a network / bridge interface
  - Sets up an IP address
  - Executes a process that you specify
  - Captures and provides application output

Hello World - Container

- In your Docker environment, just run the following command:

```
docker run busybox echo hello world
```

- We used one of the smallest, simplest images available: busybox.
- We ran a single process and echo'ed hello world.

More Useful - Container

- In your Docker environment, just run the following command:

```
docker run -it ubuntu:14.04
```

- This is a brand new container.
- It runs a bare-bones, no-frills ubuntu system.
- -i tells Docker to connect us to the container's stdin.
- -t tells Docker that we want a pseudo-terminal.

More Useful - Container

- Do something in our container:

  ls –ltr /tmp/myfile

- Let's create:

  touch /tmp/myfile

- Let's check:

  ls –ltr /tmp/myfile

More Useful - Container

- Exiting our container:

```
exit
```

- Our container is now in a stopped state.
- It still exists on disk, but all compute resources have been freed up.

Starting Another - Container

- In your Docker environment, just run the following command:

```
docker run -it ubuntu:14.04
ls –ltr /tmp/myfile
```

- We started a brand new container.

A non interactive - Container

- In your Docker environment, just run the following command:

```
docker run jpetazzo/clock
```

- This container just displays the time every second.
- This container will run forever.
- To stop it, press ^C.
- Docker has automatically downloaded the image jpetazzo/clock.

Run in background - Container

- Containers can be started in the background, with the -d flag (daemon mode):

```
docker run -d jpetazzo/clock
```

- We don't see the output of the container.
- But don't worry: Docker collects that output and logs it!
- Docker gives us the ID of the container.

List Running Containers

- With docker ps, just like the UNIX ps command, lists running processes.

```
docker ps
```

- The (truncated) ID of our container.
- The image used to start the container.
- That our container has been running (Up) for a couple of minutes.
- Now, start multiple containers and use “docker ps” to list them.

List Running Containers - Flags

- To see only the last container that was started:

```
docker ps -l
```

- To see only the ID of containers:

```
docker ps –q
```

- We can combine the flags to see ID of last container started as well.

Logs of Container

- Logs of container can be seen using:

```
docker logs 068 [[where 068 is prefix of ID]]
```

- We specified a prefix of the full container ID.
- You can, of course, specify the full ID.
- The logs command will output the entire logs of the container.
- To avoid being spammed with pages of output, we can use the –tail option:

```
docker logs --tail 3 068
```

Logs of Container

- We can also follow the logs real time using:

```
docker logs --tail 1 --follow 068
```

- This will display the last line in the log file.
- Then, it will continue to display the logs in real time.
- Use ^C to exit.

Stop our Container

- There are two ways we can terminate our detached container.
  - Killing it using the docker “kill” command.
  - Stopping it using the docker “stop” command.
- The first one stops the container immediately, by using the KILL signal.
- The second one is more graceful. It sends a TERM signal, and after 10 seconds, if the container has not stopped, it sends KILL.

Stop our Container

- Let’s stop one of our containers:

```
docker stop 47d6
```

- This might take 10 seconds.
- Docker sends the TERM signal;
- If, the container doesn't react to this signal, 10 seconds later, since the container is still running, Docker sends the KILL signal; this terminates the container.

Killing Container

- Let’s kill all our containers:

```
docker kill 068 57ad
```

- The stop and kill commands can take multiple container IDs.
- Those containers will be terminated immediately (without the 10 seconds delay).
- Do check running containers now.

List stopped Container

- Let’s stopped containers:

```
docker ps -a
```

Removing Container

- Let’s remove our container:

```
docker rm <yourContainerID>
```

Debugging a Container

- You can SSH into a container using “docker exec”:

```
docker exec -it <yourContainerId> bash
```

Default Names - Containers

- When we create a container, if we don't give a specific name, Docker will pick one for us.
- It will be the concatenation of:
  - A mood (furious, goofy, suspicious, boring...)
  - The name of a famous inventor (tesla, darwin, wozniak...)
- Examples: happy_curie, clever_hopper, jovial_lovelace …

Specify Name - Containers

- You can set the name of the container when you create it.

```
docker run --name ticktock jpetazzo/clock
```

- If you specify a name that already exists, Docker will refuse to create the container.
- You can rename containers with “docker rename”.
- This allows you to "free up" a name without destroying the associated container.

Use Case - Containers

- Run the Docker Hub image nginx, which contains a basic web server:

```
docker run -d -P nginx
```

- Docker will download the image from the Docker Hub.
- -d tells Docker to run the image in the background.
- -P tells Docker to make this service reachable from other computers. (-P is the short version of --publish-all.)

Use Case - Containers

- Finding our WebServer Port:

  ```
  docker ps
  ```

- Connecting to our web server (GUI):

  ```
  http://dockerHostIP:Port
  ```

Use Case - Containers

- If you want to set port numbers yourself, no problem:

```
docker run -d -p 8000:80 nginx
```

- We are running two NGINX web servers.
- The first one is exposed on port 80.
- The second one is exposed on port 8000.
- The convention is port-on-host:port-on-container.

Session: 7

DevOps - Terraform

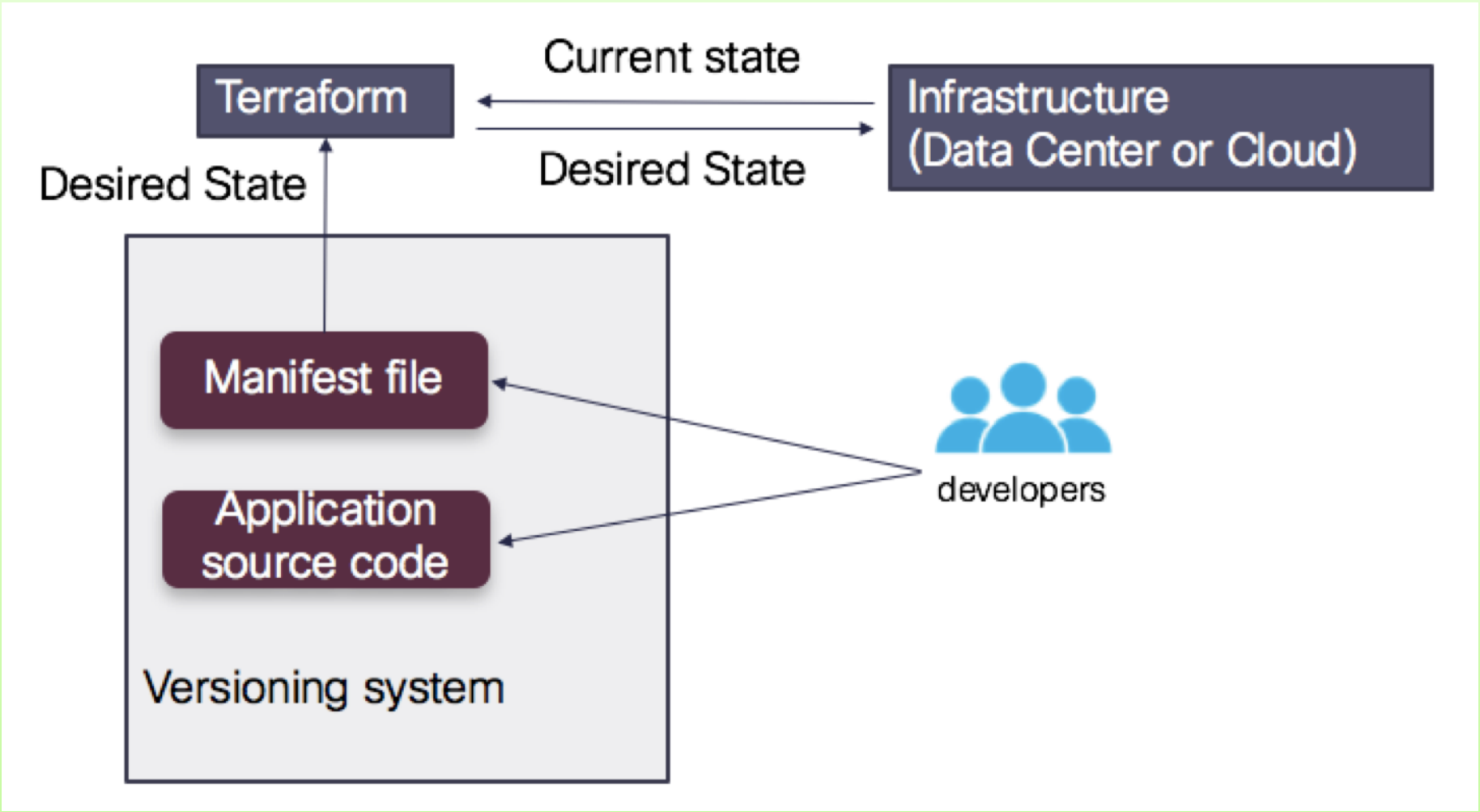
What is Terraform?

- Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently.
- Configuration files describe to Terraform the components needed to run a single application or your entire datacenter.
- Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure.
- As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.
- The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

Features of Terraform

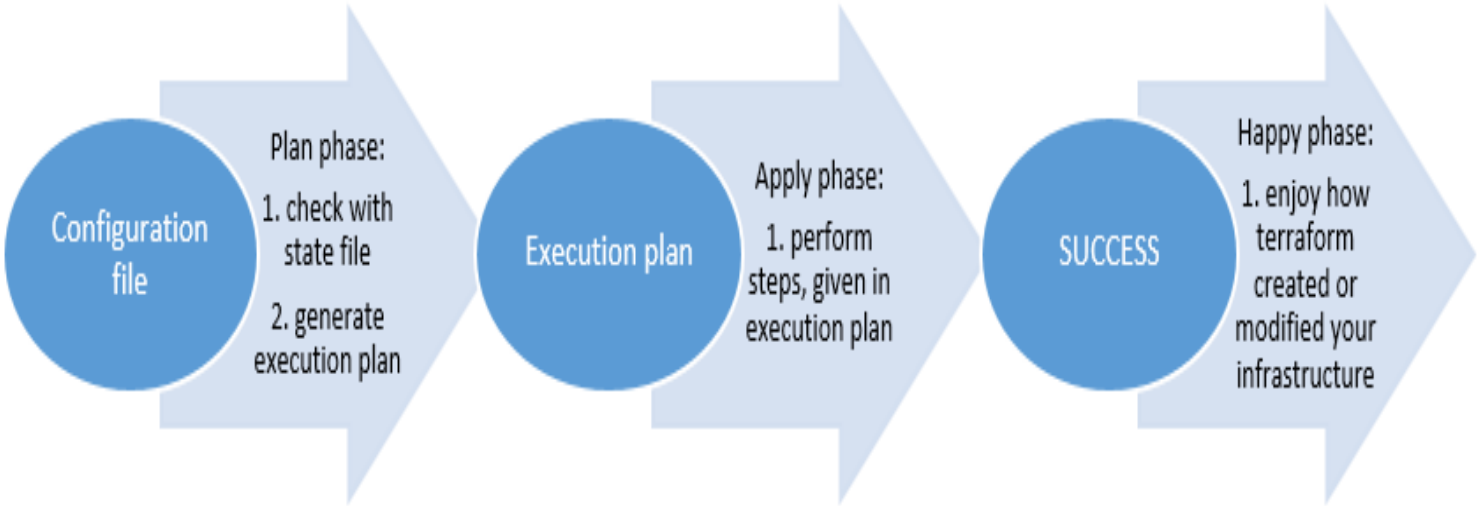
- Infrastructure as Code(IaC)
- Execution Plans
- Resource Graph
- Change Automation
- Collaboration

How Terraform works?



Atgen

How Terraform works?



Atgen

Terraform – Lab Setup

- Terraform will be installed for Lab Environment.

Terraform – Lab Setup

- Get Zip Terraform Package using wget:

  wget https://releases.hashicorp.com/terraform/0.13.5/terraform_0.13.5_linux_amd64.zip

- Unzip the Package:

  unzip terraform_0.13.5_linux_amd64.zip

- Share on server:

  cp terraform /usr/bin/

Terraform – Configuration

- Terraform uses text files to describe infrastructure and to set variables. These text files are called Terraform *configurations* and end in .tf.
- The format of the configuration files are able to be in two formats: Terraform format and JSON.
- The Terraform format is more human-readable, supports comments, and is the generally recommended format for most Terraform files.
- The JSON format is meant for machines to create, modify, and update.

Load Order & Semantics

- When invoking any command that loads the Terraform configuration, Terraform loads all configuration files within the directory specified in alphabetical order.
- The files loaded must end in either .tf or .tf.json to specify the format that is in use. Otherwise, the files are ignored.
- Override files are the exception, as they're loaded after all non-override files, in alphabetical order.
- The configuration within the loaded files are appended to each other.
- The order of variables, resources, etc. defined within the configuration doesn't matter.

Configuration Syntax

- The syntax of Terraform configurations is called HashiCorp Configuration Language (HCL).
- It is meant to strike a balance between human readable and editable as well as being machine-friendly.
- For machine-friendliness, Terraform can also read JSON configurations.
- For general Terraform configurations, however, we recommend using the HCL Terraform syntax.

Configuration Syntax

```
# An AMI
variable "ami" {
  description = "the AMI to use"
}
/* A multi
   line comment. */
resource "aws_instance" "web" {
  ami                  = "${var.ami}"
  count                = 2
  source_dest_check = false
  connection {
    user = "root"
  }
}
```

Providers

- Terraform is used to create, manage, and update infrastructure resources such as physical machines, VMs, network switches, containers, and more.
- Almost any infrastructure type can be represented as a resource in Terraform.
- A provider is responsible for understanding API interactions and exposing resources.
- Providers generally are an IaaS (e.g. AWS, GCP, Microsoft Azure, OpenStack), PaaS (e.g. Heroku), or SaaS services (e.g. Terraform Enterprise, DNSimple, CloudFlare).

Providers

- All the terraform providers can be found at:

https://www.terraform.io/docs/providers/index.html

Provider Configurations

- Providers are responsible in Terraform for managing the lifecycle of a resource: create, read, update, delete.
- Most providers require some sort of configuration to provide authentication information, endpoint URLs, etc.
- By default, resources are matched with provider configurations by matching the start of the resource name. For example, a resource of type vsphere_virtual_machine is associated with a provider called vsphere.

Provider Configurations - Example

- A provider configuration looks like the following:

```
provider "aws" {
    access_key = "foo"
    secret_key = "bar"
    region     = "us-east-1"
}
```

Provider Initializations

- Each time a new provider is added to configuration, it's necessary to initialize that provider before use.
- Initialization downloads and installs the provider's plugin and prepares it to be used.
- Provider initialization is one of the actions of terraform init. Running this command will download and initialize any providers that are not already initialized.
- Providers downloaded by terraform init are only installed for the current working directory.
- Note that terraform init cannot automatically download providers that are not distributed by HashiCorp.

Multiple Provider Instances

- You can define multiple configurations for the same provider in order to support multiple regions, multiple hosts, etc.

```
# The default provider configuration
      provider "aws" {
          # ...
      }
# Additional provider configuration for west coast region
      provider "aws" {
          alias  = "west"
          region = "us-west-2"
      }
```

Multiple Provider Instances

- Using provider in resource:

```
resource "aws_instance" "foo" {
    provider = "aws.west"
    # ...
}
```

Resource Configurations

- The most important thing you'll configure with Terraform are resources.
- Resources are a component of your infrastructure.
- It might be some low level component such as a physical server, virtual machine, or container. Or it can be a higher level component such as an email provider, DNS record, or database provider.
- The resource block creates a resource of the given TYPE (first parameter) and NAME (second parameter). The combination of the type and name must be unique.

Resource Configurations - Example

- A resource configuration looks like the following:

```
resource "aws_instance" "web" {
    ami           = "ami-408c7f28"
    instance_type = "t1.micro"
}
```

Terraform Commands (CLI)

- Terraform is controlled via a very easy to use command-line interface (CLI).
- Terraform is only a single command-line application: terraform. This application then takes a subcommand such as "apply" or "plan".
- The terraform CLI is a well-behaved command line application. In erroneous cases, a non-zero exit status will be returned. It also responds to -h and --help as you'd most likely expect.

terraform --help

Terraform Commands (CLI)

- Get Terraform plugins as per configuration:

  terraform init

- Validate Terraform configurations:

  terraform validate

- Validate configurations in Simulation mode:

  terraform plan

- Apply Terraform configurations:

  terraform apply

Terraform Commands (CLI)

- Destroy Terraform configuration:

  terraform destroy

- Save a plan:

  terraform plan -out=./plan

- Apply a plan:

  terraform apply ./plan

- Show plan or state:

  terraform show

Exercise

- Create an AWS EC2 instance using terraform.

Exercise

- Create a Docker Container of NGINX with service exposed on port 1000 using terraform.

- Hint: Installation of Docker
  - yum install docker
  - systemctl restart docker
  - systemctl enable docker

Session: 8

Kubernetes

What is Kubernetes

- Began in 2014 by Google, inspired by their Borg and other internal container scaling projects.
- It has quickly risen to dominate the container orchestration space.
- Heavyweights behind it including Red Hat, IBM.

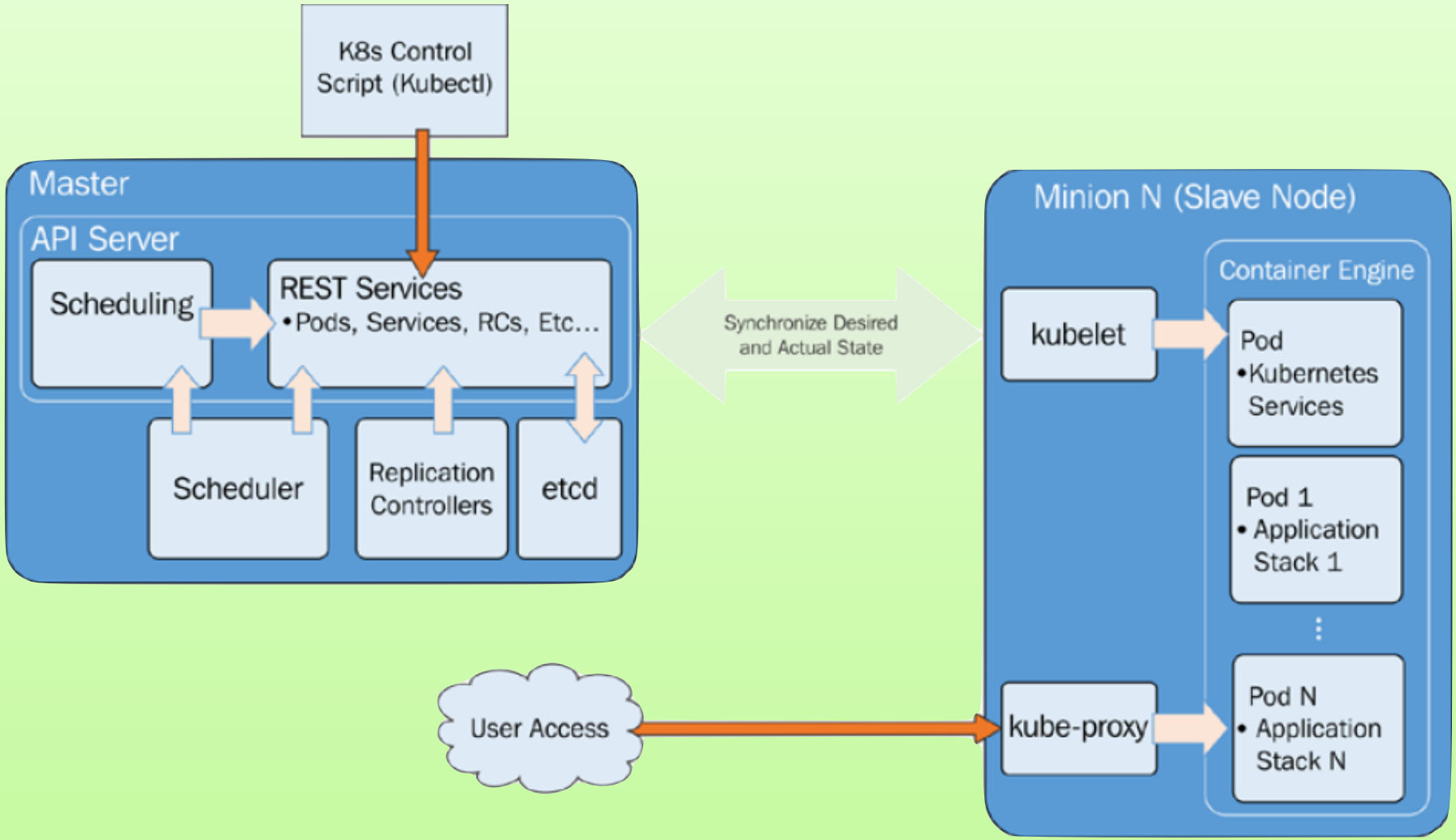
What is Kubernetes

- Kubernetes is an open-source container management (orchestration) tool. It's container management responsibilities include container deployment, scaling & descaling of containers & container load balancing.
- Kubernetes is not a containerisation platform. It is a multi-container management solution.
- Kubernetes is a portable, extensible open-source platform for managing containerised workloads and services, that facilitates both declarative configuration and automation.
- Google open-sourced the Kubernetes project in 2014.
- Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale, combined with best-of-breed ideas and practices from the community.

What is Kubernetes

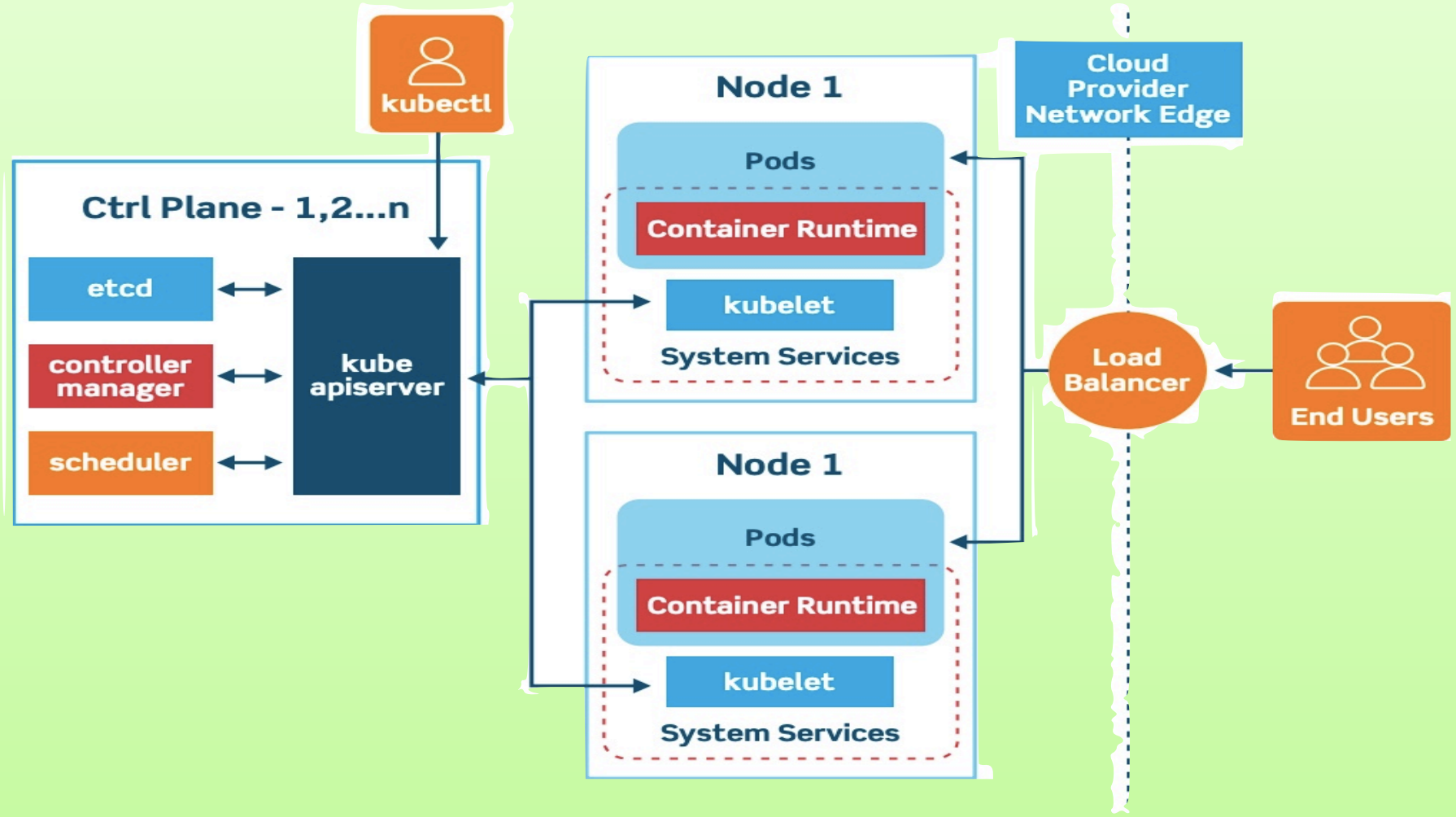
- Kubernetes brings assistance to orchestrating containers at scale as well as managing full application stacks.
- K8s moves up the stack giving us constructs to deal with management at the application or service level.
- This gives us automation and tooling to ensure high availability, application stack, and service-wide portability.
- K8s also allows finer control of resource usage, such as CPU, memory, and disk space across our infrastructure.
- Kubernetes provides this higher level of orchestration management by giving us key constructs to combine multiple containers, endpoints, and data into full application stacks and services.
- K8s then provides the tooling to manage the when, where, and how many of the stack and its components.

Kubernetes – Core Architecture



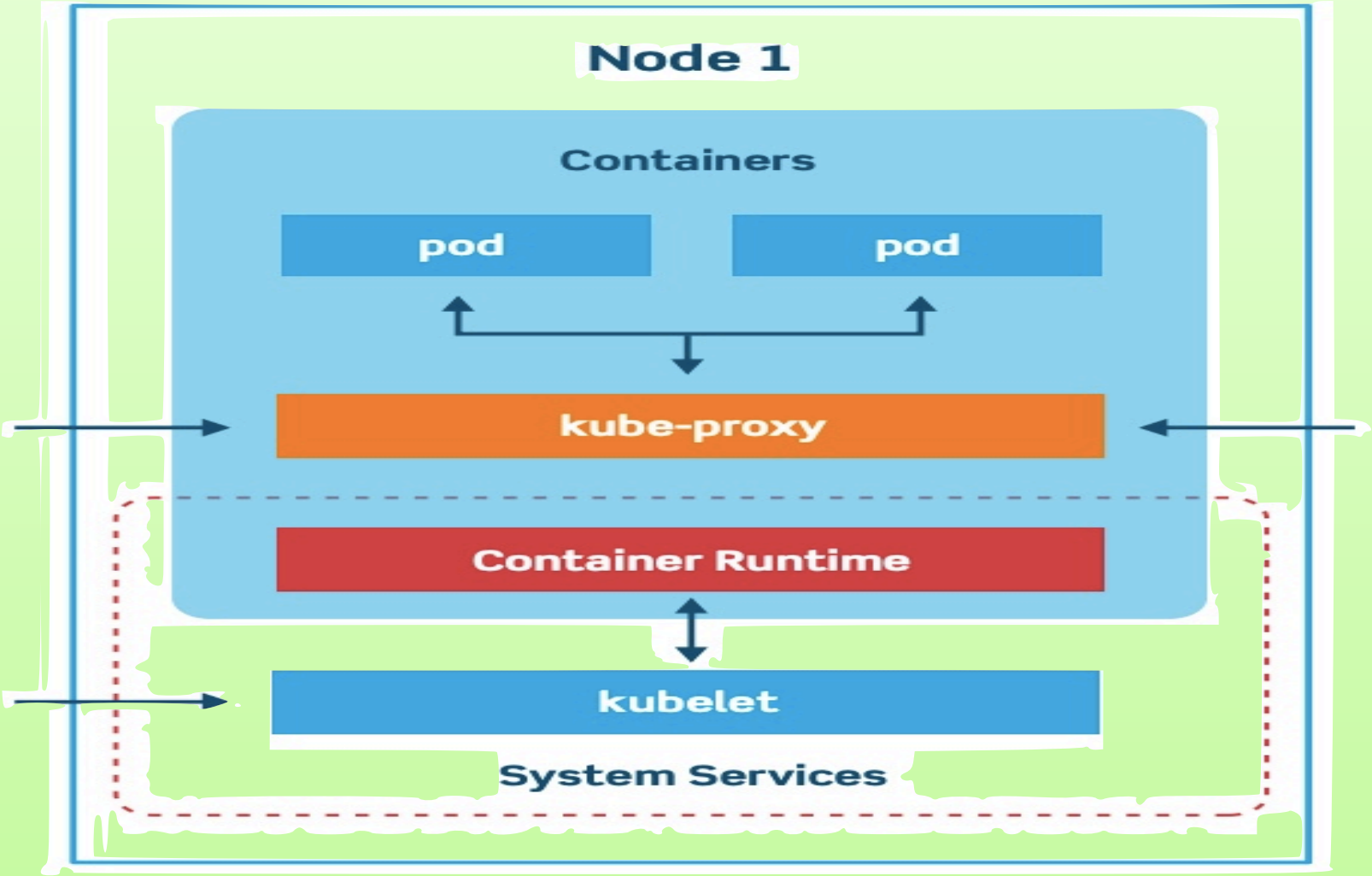
Atgen

Kubernetes - Core Architecture



Atgen

Kubernetes - Cluster Nodes



Atgen

Kubernetes - Master

- MASTER is the brain of our cluster
- Following are key components in master:
  - API Server
  - Scheduler
  - Replication Controller
  - etcd

Master – API Server

- API server maintains RESTful web services for querying and defining our desired cluster and workload state.
- It's important to note that the control pane only accesses the master to initiate changes and not the nodes directly.

Master - Scheduler

- Scheduler works with the API server to schedule workloads in the form of pods on the actual minion nodes.
- These pods include the various containers that make up our application stacks.
- By default, the basic Kubernetes scheduler spreads pods across the clusters and uses different nodes for matching pod replicas.
- Kubernetes also allows specifying necessary resources for each container, so scheduling can be altered by these additional factors.

Master – Replication Controller

- The replication controller works with the API server to ensure that the correct number of pod replicas are running at any given time.
- This is exemplary of the desired state concept. If our replication controller is defining three replicas and our actual state is two copies of the pod running, then the scheduler will be invoked to add a third pod somewhere on our cluster.
- The same is true if there are too many pods running in the cluster at any given time.
- In this way, K8s is always pushing towards that desired state.

Master – etcd

- ‘etcd’ running as a distributed configuration store.
- The Kubernetes state is stored here and ‘etcd’ allows values to be watched for changes.
- Think of this as the brain's shared memory.

Node (formerly minions)

- In each node, we have a couple of components.
- The ‘kubelet’ interacts with the API server to update state and to start new workloads that have been invoked by the scheduler.
- ‘kube-proxy’ provides basic load balancing and directs traffic destined for specific services to the proper pod.
- We also have some default pods, which run various infrastructure services for the node. These pods include services for Domain Name System (DNS), logging, and pod health checks. The default pod will run alongside our scheduled pods on every node.

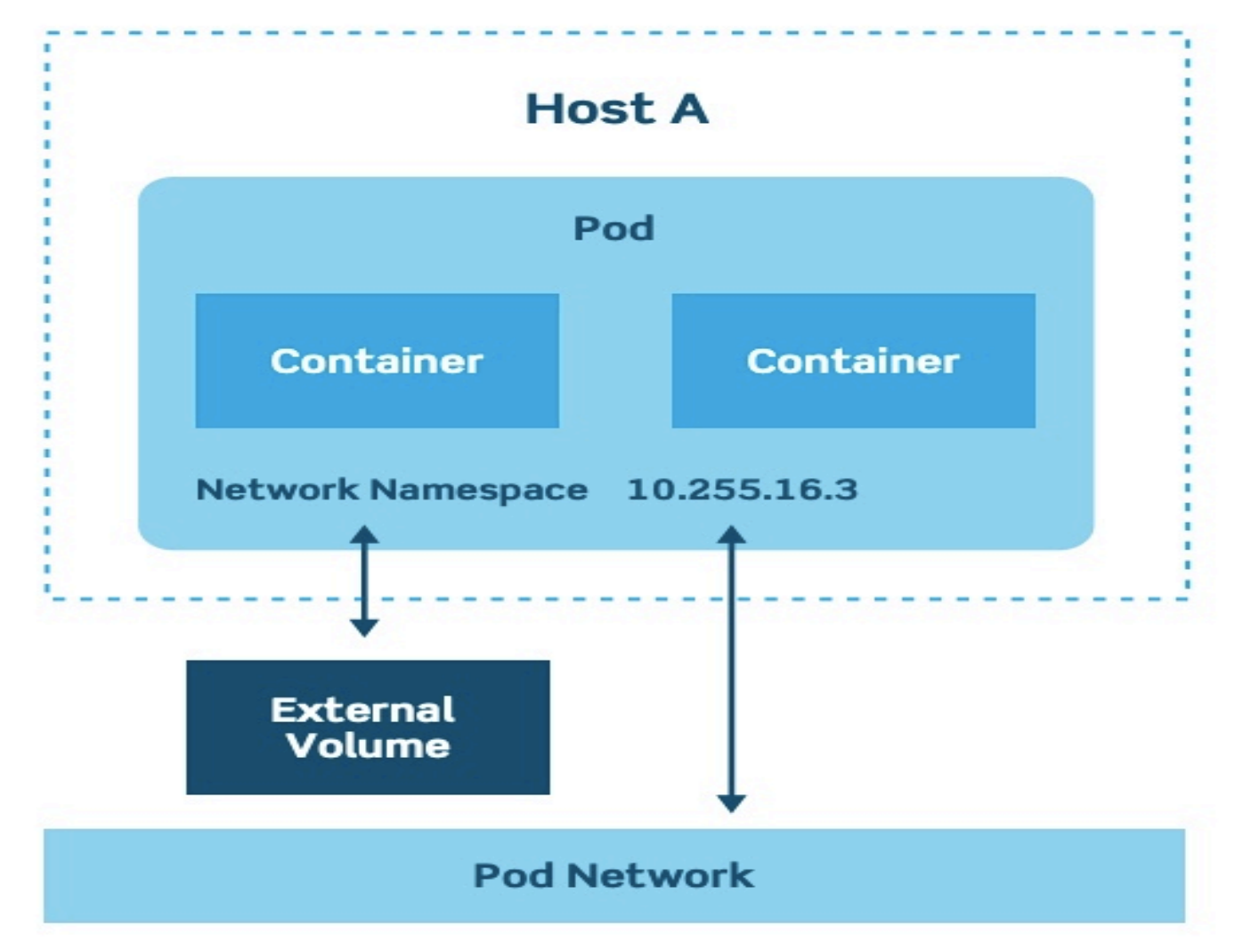
LAB Environment

- In this section we are going to do following:
  - Install Kubeadm/Docker on master/minion
  - Setup K8s Cluster
  - Join K8s Cluster
  - Configure kubectl
  - Validate Installation

Kubernetes – A ‘pod’

- Pods allow you to keep related containers close in terms of the network and hardware infrastructure.
- Data can live near the application, so processing can be done without incurring a high latency from network traversal.
- Similarly, common data can be stored on volumes that are shared between a number of containers.
- Pods essentially allow you to logically group containers and pieces of our application stacks together.
- While pods may run one or more containers inside, the pod itself may be one of many that is running on a Kubernetes (minion) node.
- So, pods give us a logical group of containers that we can then replicate, schedule, and balance service endpoints across.

Kubernetes – ‘pod’



Atgen

‘pod’ - Example

- Create a file ‘nodejs-pod.yaml’ with below contents:

```
apiVersion: v1
kind: Pod
metadata:
  name: node-js-pod
spec:
  containers:
  - name: node-js-pod
    image: bitnami/apache:latest
    ports:
    - containerPort: 80
```

‘pod’ - Example

- Create a pod using following command:

```
kubectl create -f nodejs-pod.yaml
```

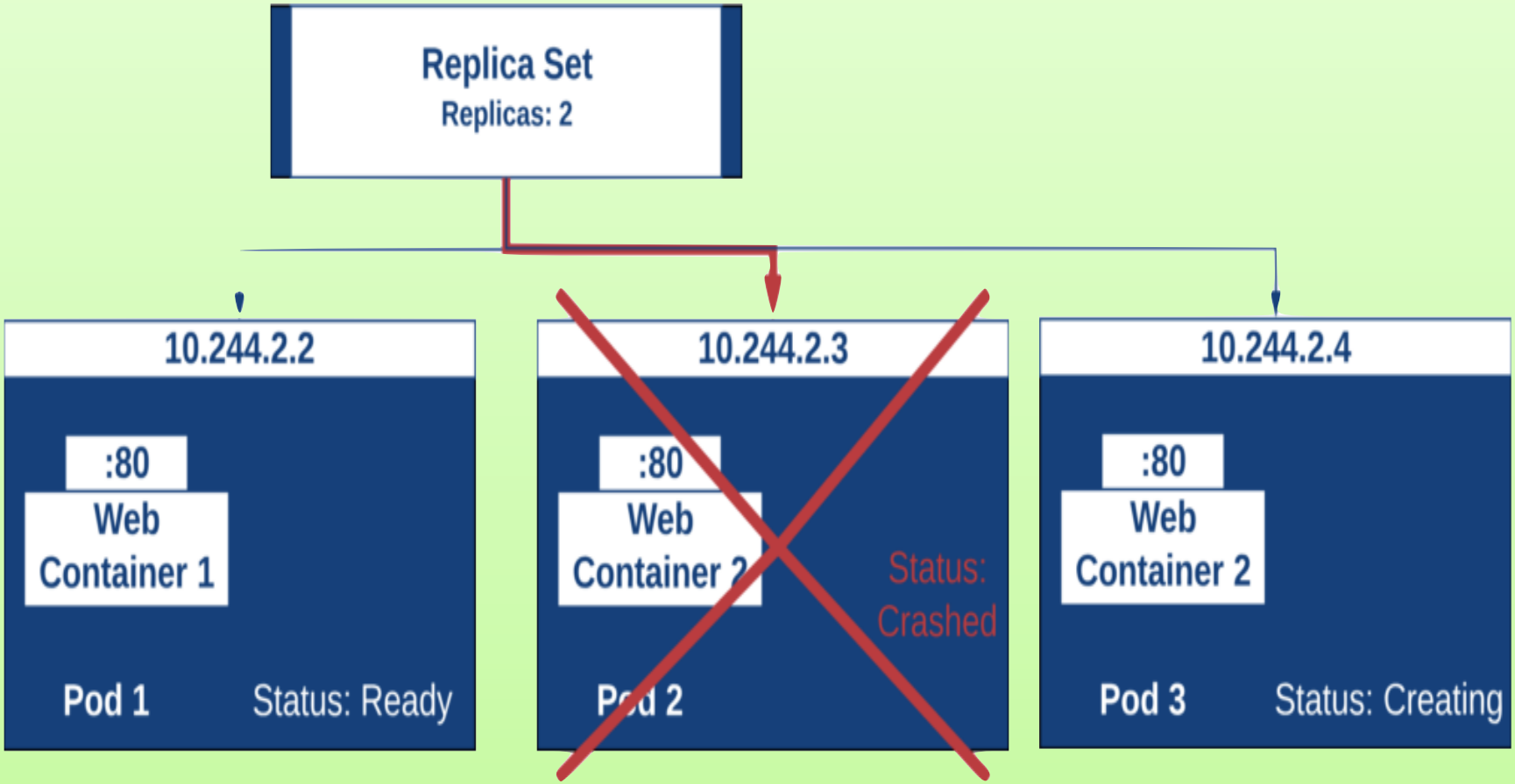
- See details about pod:

```
kubectl describe pods/node-js-pod
```

Kubernetes – ReplicaSets

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.
- A ReplicaSet is defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria.
- A ReplicaSet then fulfills its purpose by creating and deleting Pods as needed to reach the desired number.
- When a ReplicaSet needs to create new Pods, it uses its Pod template.

Kubernetes – ReplicaSets



Atgen

ReplicaSets - Example

- Create a file ‘frontend-rs.yaml’:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

Kubernetes – Deployment

- A Deployment controller provides declarative updates for Pods and ReplicaSets.
- Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

Deployment - Example

- Create a file ‘nginx-deployment.yaml’:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Deployment - Example

- Create a deployment using command:

```
kubectl create -f nginx-deployment.yaml
```

- See details about pod:

```
kubectl get deployments
```

Deployment - Example

- Now, update nginx image to ‘1.9.1’ version:

```
kubectl edit deployment/nginx-deployment
```

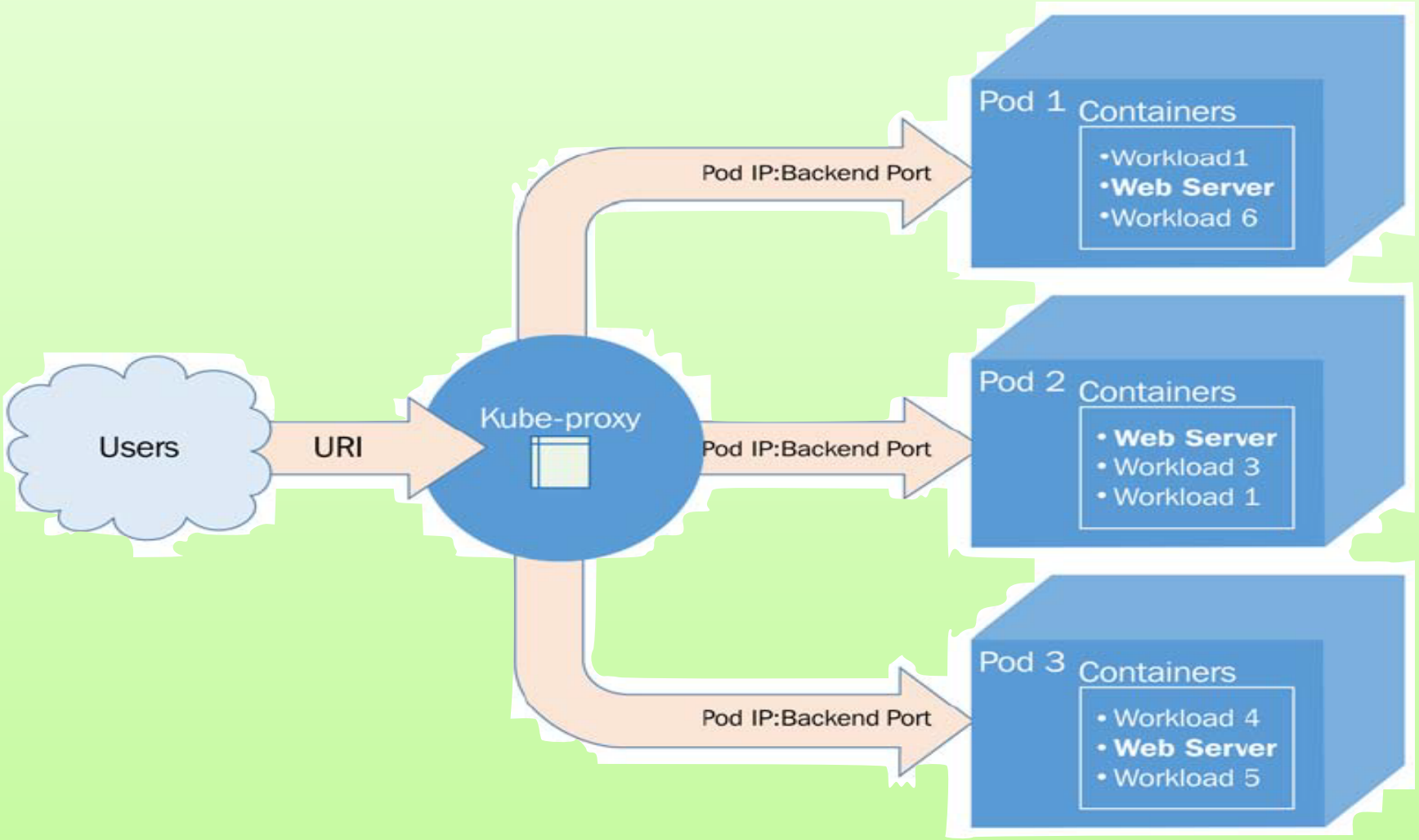
- To see rollout status:

```
kubectl rollout status deployment/nginx-deployment
```

Kubernetes – Services

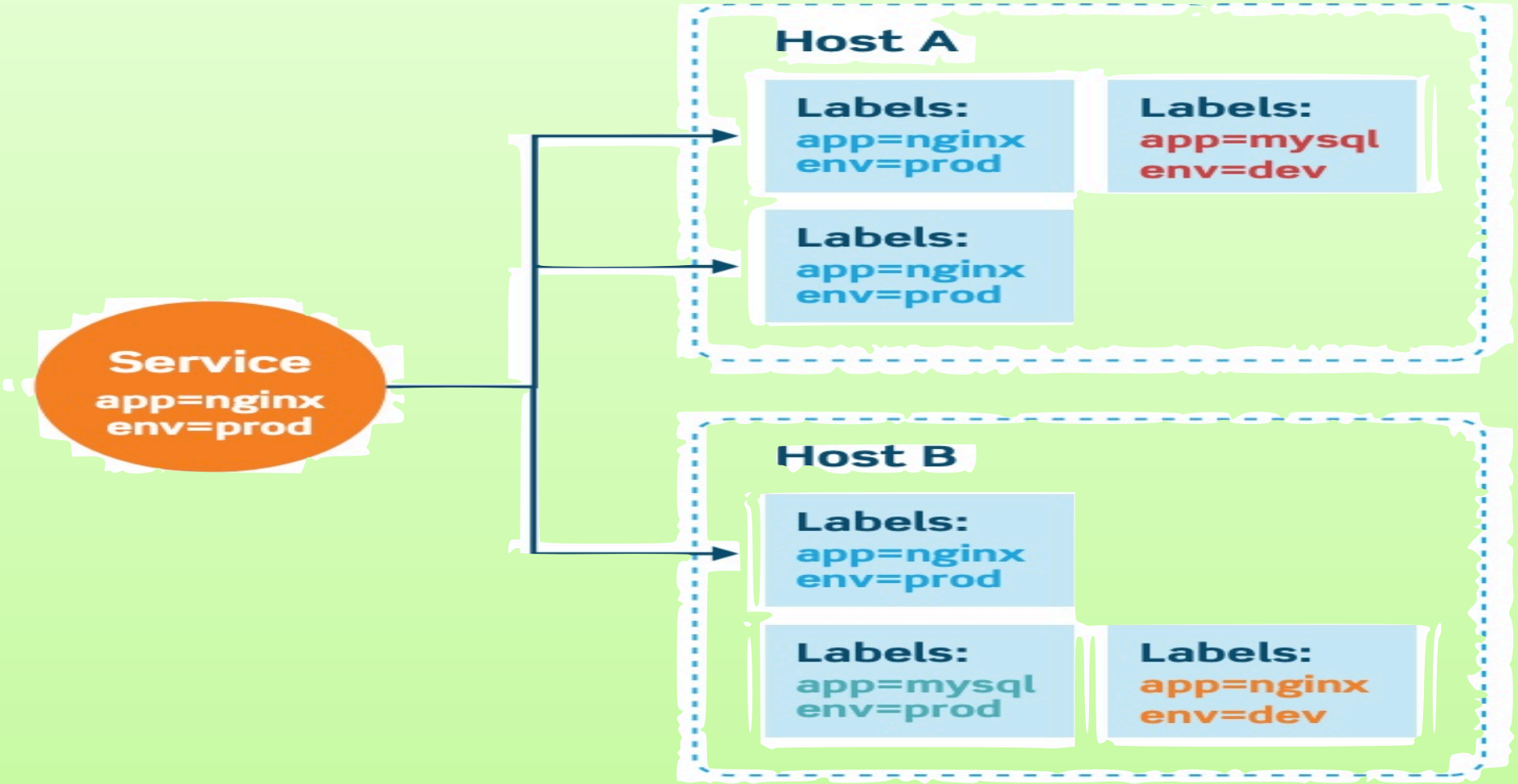
- Services allow us to abstract access away from the consumers of our applications.
- Using a reliable endpoint, users and other programs can access pods running on your cluster seamlessly.
- K8s achieves this by making sure that every node in the cluster runs a proxy named kube-proxy. As the name suggests, kube-proxy's job is to proxy communication from a service endpoint back to the corresponding pod that is running the actual application.
- A virtual IP address and port are used as the entry point for the service, and traffic is then forwarded to a random pod on a target port defined.
- Service definitions are monitored and coordinated from the K8s cluster master and propagated to the kube-proxy daemons running on each node.

Kubernetes - Services



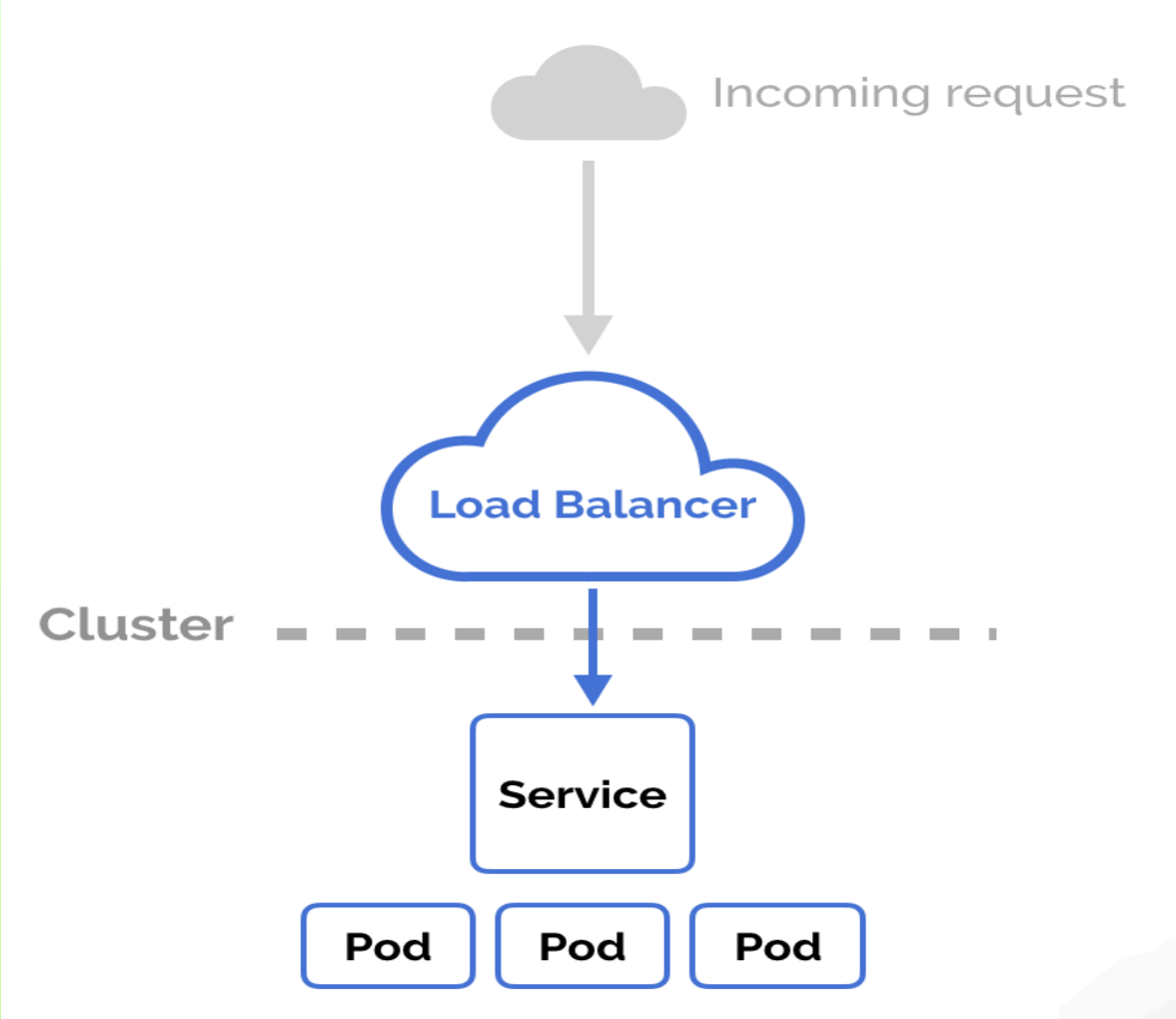
Atgen

Kubernetes - Services



Atgen

Services - LoadBalancer



Atgen

Kubernetes Service - Example

- Create a file ‘nginx-service.yaml ‘ with below contents:

```
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
```

Kubernetes Application - Example

- Create the Deployment for Nginx:

```
kubectl create -f nginx-deployment.yaml
```

- Create the NGINX service:

```
kubectl create -f nginx-service.yaml
```

Kubernetes Application - Example

- Let’s look at running services:

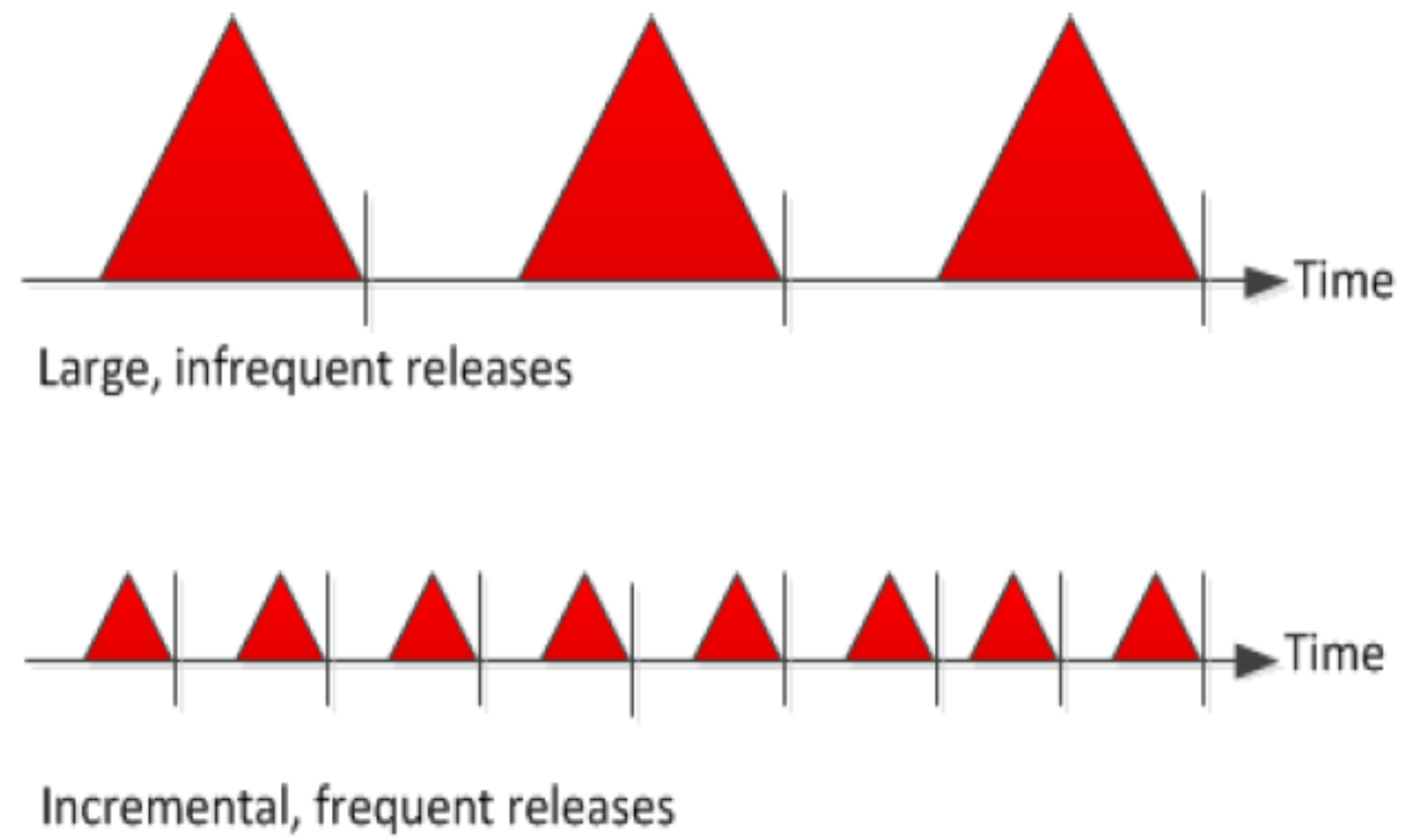
```
kubectl get services
```

- Now, see if we can connect by opening up the public address in a browser.

Session: 9

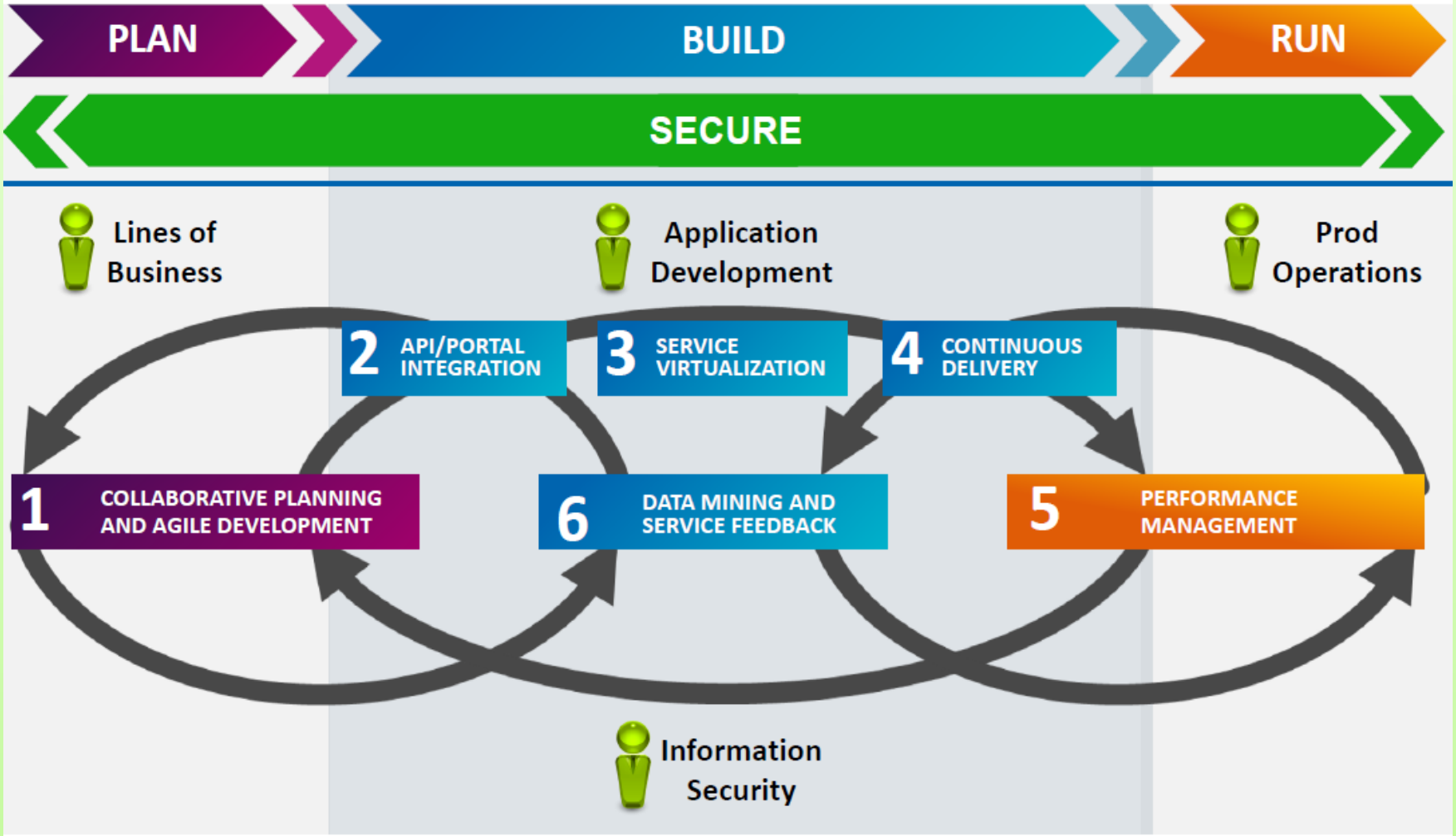
DevOps Application Delivery

Releases - Changing Patteren



Atgen

DevOps – Application Delivery LifeCycle



Atgen

Application Delivery Pipeline

- To understand application delivery in concrete terms, let's look in more detail at the component technologies that are required to implement it.
  - Version Control System
  - Automated Build System
  - Package Repository
  - CI & CD
  - Automated Testing
  - Infrastructure as Code

Version Control System

- Version-control systems allow you to keep multiple versions of files so that when you modify something you can still access previous versions.
- A good version-control system stores and gives access to all the files that have ever been stored.
- It also allows people who may be distributed across the world to collaborate.
- Examples of popular source-control systems include GitHub's Git, Apache's Subversion, and Microsoft's Team Foundation Server.

Automated Build System

- An automated build system allows you to write scripts that describe the entire build process.
- You may specify the sequence of tasks in the build process, prepare a build environment, validate the source code, compile the sources, run system tests, build the documentation, produce packages that contain all the artifacts, and release builds to a package repository.
- Popular tools include Apache Maven, Ant, Gradle, uBuild, and Microsoft Team Build, which is part of Microsoft Visual Studio.

Package Repository

- A package contains everything necessary to install your application as well as information such as the package's version, documentation, data, and configuration information.
- A package repository stores packages that you want to make available for deployment.
- Package-management systems (or repositories) allow you to perform tasks such as installing, uninstalling, verifying, querying, and updating software packages.

Package Repository

- By providing an authoritative source for release artifacts, a well-managed package repository can help address issues of compliance, traceability, and audit while enabling releases to proceed more quickly.
- A good package repository can enable faster build times, facilitate more-efficient collaboration among teams, and reduce the learning curve for new developers.

CI & CD Server

- Continuous integration (CI) means that every time someone commits a change, the entire application or environment is rebuilt and a series of automated tests are run against it.
- Continuous integration requires version control, automated build, and commitment from the team.
- Continuous integration requires:
  - Version Control
  - Automated Build
  - Commitment from the team

Automated Testing

- Automated test suites should test the functional aspects of the system as well as non-functional requirements such as capacity and security.
- Testing involves the entire team and should be done continuously from the beginning of the project.
- The goal is to catch problems early, when they are easiest to fix.
- Automated tests should include Unit Tests, Functional tests, and build-verification tests (BVTs).

Automated Testing

- Benefits of automated testing include:
  - Developers get quick feedback on their work.
  - Testers do not need to perform repetitive tasks and can concentrate on more-complex activities.
  - Tests can be used as a regression test suite to prevent the reintroduction of previous defects.
  - Tests can be used to automatically generate requirements documentation.

Infrastructure as Code

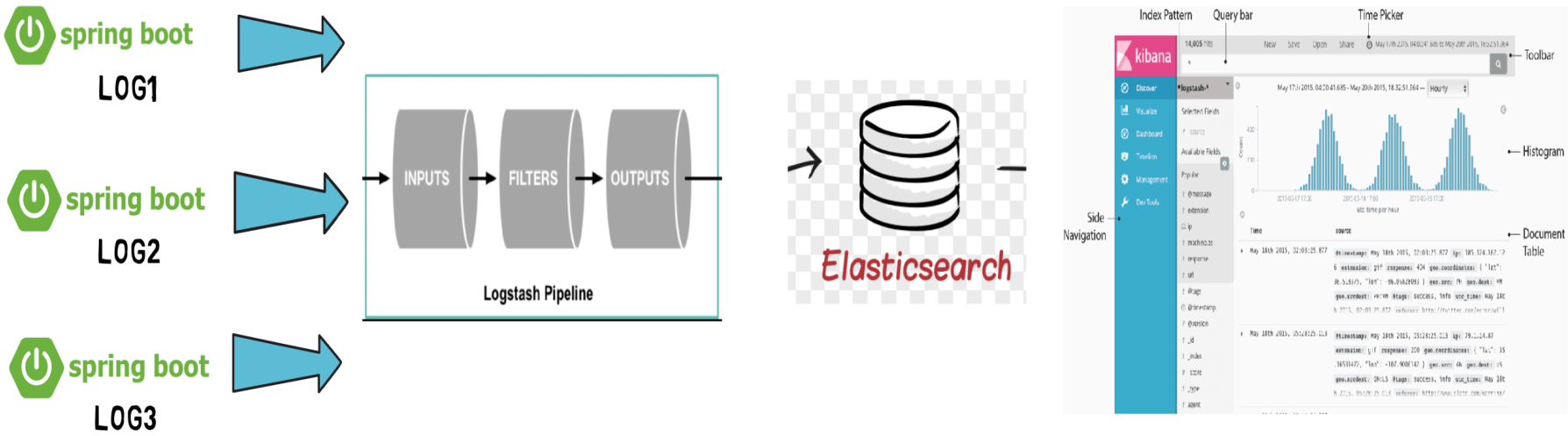
- The most obvious point about virtually any company's infrastructure is that it has grown increasingly complex over the years. People are constantly looking for ways to improve scalability and manageability. You need:
  - Virtualization, which allows isolation, dynamism, and better resource utilization and is most often the underpinning of cloud computing
  - Cloud computing, which provides scalable computing resources and a pay-as-you-go model
  - Continuous delivery, where infrastructure is treated as code and goes through the same automated pipeline as application source code

Exercise

- Install & Start Web Application on App Server with all pre-requisites.

ELK - Stack

ELK - ELASTIC SEARCH - LOGSTASH - KIBANA - ARCHITECTURE



MICROSERVICES LOGSTASH ELASTICSEARCH KIBANA

Atgen

Contact us

- Contact @ http://www.atgensoft.com/
- Linkedin: @atgenautomation
- Twitter: @atgenautomation
- FaceBook: @atgenautomation
- Youtube: @atgenautomation
- Email: SAGAR.MEHTA@ATGENSOFT.COM

Thank You !!

Atgen