Site Reliability Engineering

Overview

Agenda

- What is SRE?
- Why SRE?
- SRE - Service Labels
- SRE - Error Budgets
- SRE - Toil
- SRE – Readiness
- SRE – IaC
- SRE - Containerization
- SRE - Observability

Some Assumptions

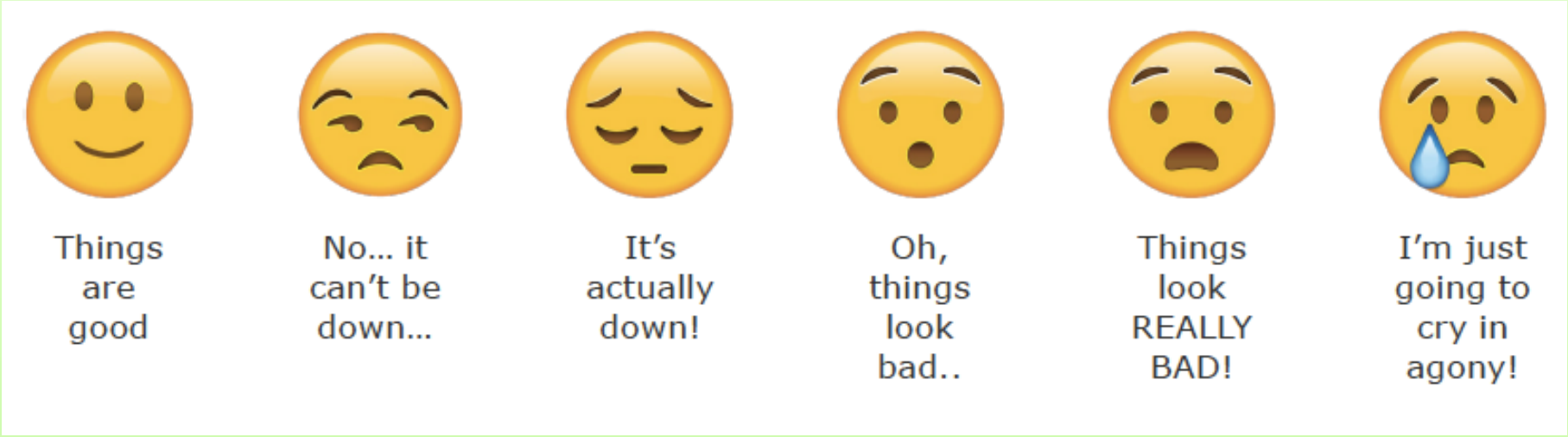
- SRE, by definition, is never a “one-size-fits-all” remedy
- SRE is about drilling down on your own organisation’s specific problems and challenges

Session: 1

Site Reliability Engineering

State transitions during an outage…

@3:32 am



ATGEN

State transitions during an outage…

Let’s shift operations from

reactive

to

proactive

Topics

Introduction to SRE

Enabling SREs

Budgeting for Failures

Measuring Systems

Managing Incidents

Technology Enablers

ATGEN

Introduction to SRE

Typical Software Lifecycle

Sanity Check

- The goal is not just for software to be pushed out, but to also be maintained once its live.
- How long does it take for an incident to reach the right team?
  - Can it be minutes, instead of hours?
- How often does the same incident re-occur?
  - Can it be reduced to never?

Reliability is the outcome of a system to be able to withstand some certain types of failure and still remain functional from a user's perspective.

Site Reliability Engineering

- SRE is what happens when a software engineer is put to solve operations problems.
- Site reliability engineering (SRE) is a specialised skillset where software engineers develop and contribute to the ongoing operation of systems in production.

Site Reliability Engineering

Site Reliability Engineer

Spend 50% of contributions in Systems Engineering problems

--

infrastructure, operating system & network, databases

--

availability, observability, efficiency & reliability of services

Spend 50% of contributions in Software Engineering problems

--

application design, data structure, service design & key integrations

--

development tasks that contribute to systems reliability

ATGEN

SRE approach to Operations

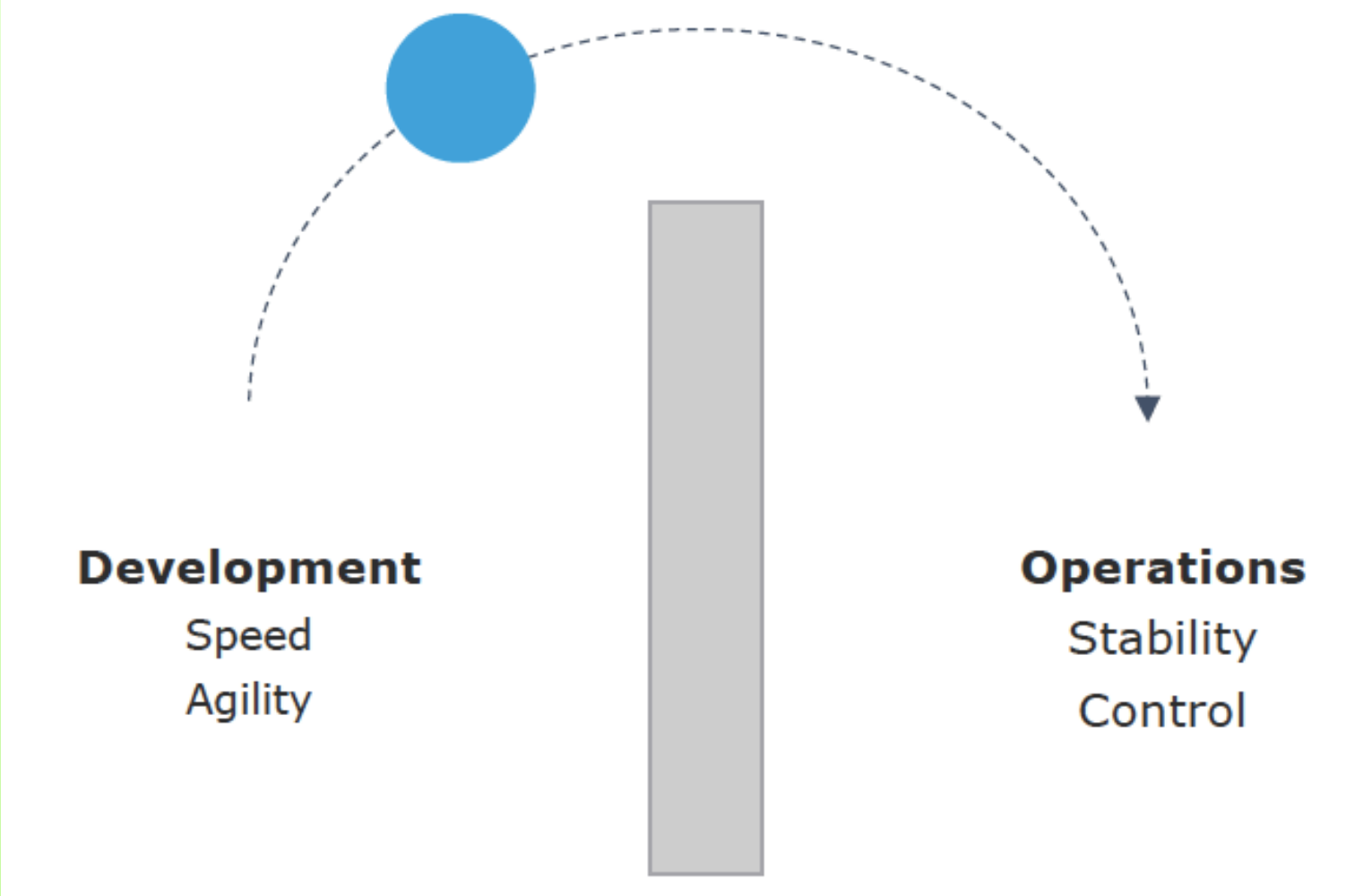
- Treat operations like a software engineering problem.
- Automate tasks normally done by system administers (such as deployments).
- Design more reliable and operable service architectures from the ground-up, not an afterthought.

Why another discipline

- The evolution of systems requires an evolution of systems engineers.
- Common Issues:
  - Reliability is usually left to the architects to design, typically at the beginning of a product design.
  - Non-functional requirements are not reviewed as often.
  - Change of functionality may impact previously assumed reliability requirements.
  - Most outages are typically caused by change, and a typical Ops mindset does not necessarily prepare for fast remedy.

In the new world, think of systems as software-defined.

Wall of Confusion



ATGEN

SRE implements DevOps

DevOps

- A set of principles & culture guidelines that helps to breakdown the silos between development and operations / networking / security

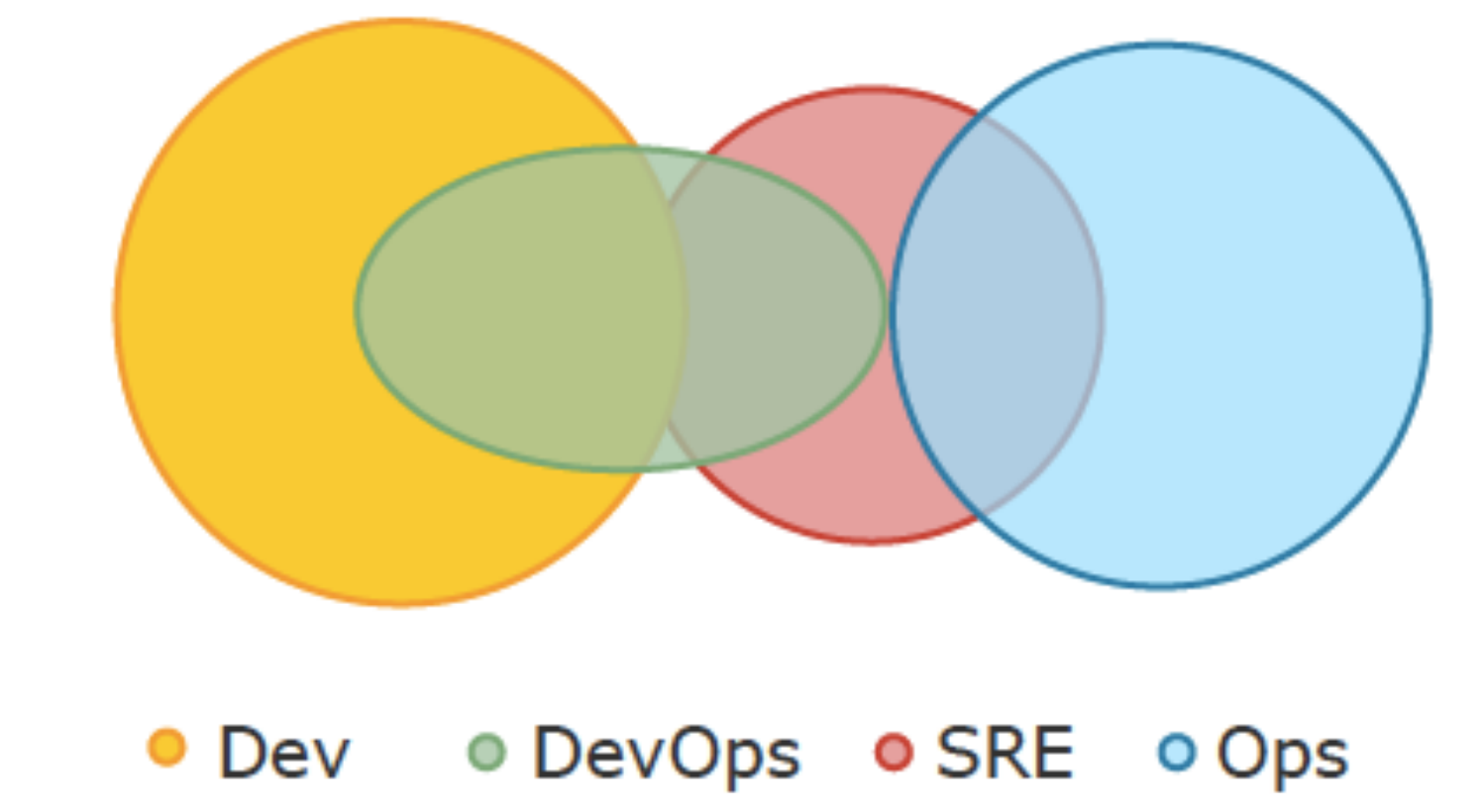
- Reduce Silos
- Plan for failure
- Small batch changes
- Tooling & automation
- Measure everything

SRE

- A set of practices with an emphasis of strong engineering capabilities that implement the DevOps practices, and sets a job role + team

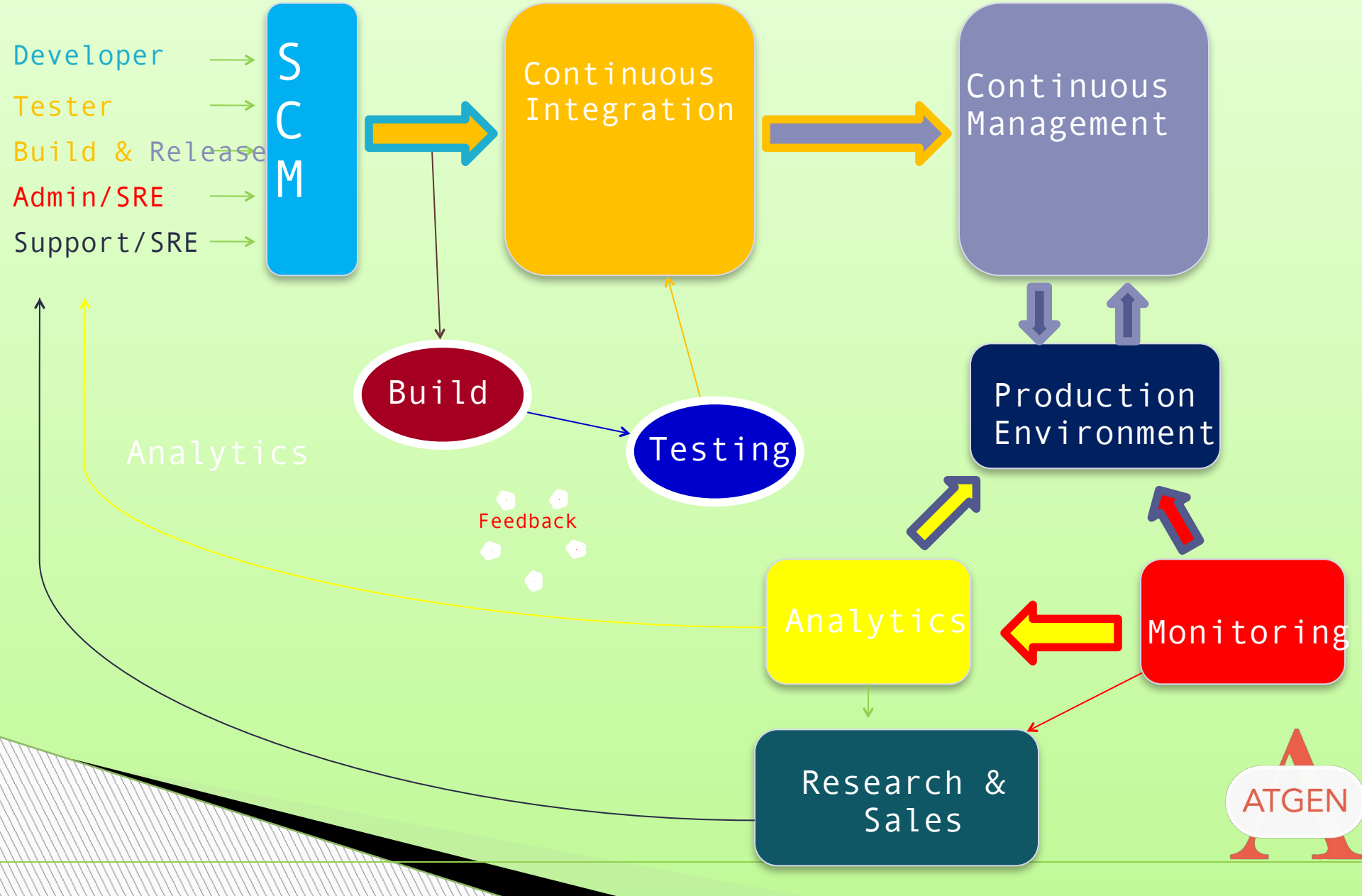
- Reduce Silos by shared ownership
- Plan for failure using error budgets
- Small batch changes with focus on stability
- Tooling & automation of manual tasks
- Measure everything monitoring the right things

SRE implements DevOps

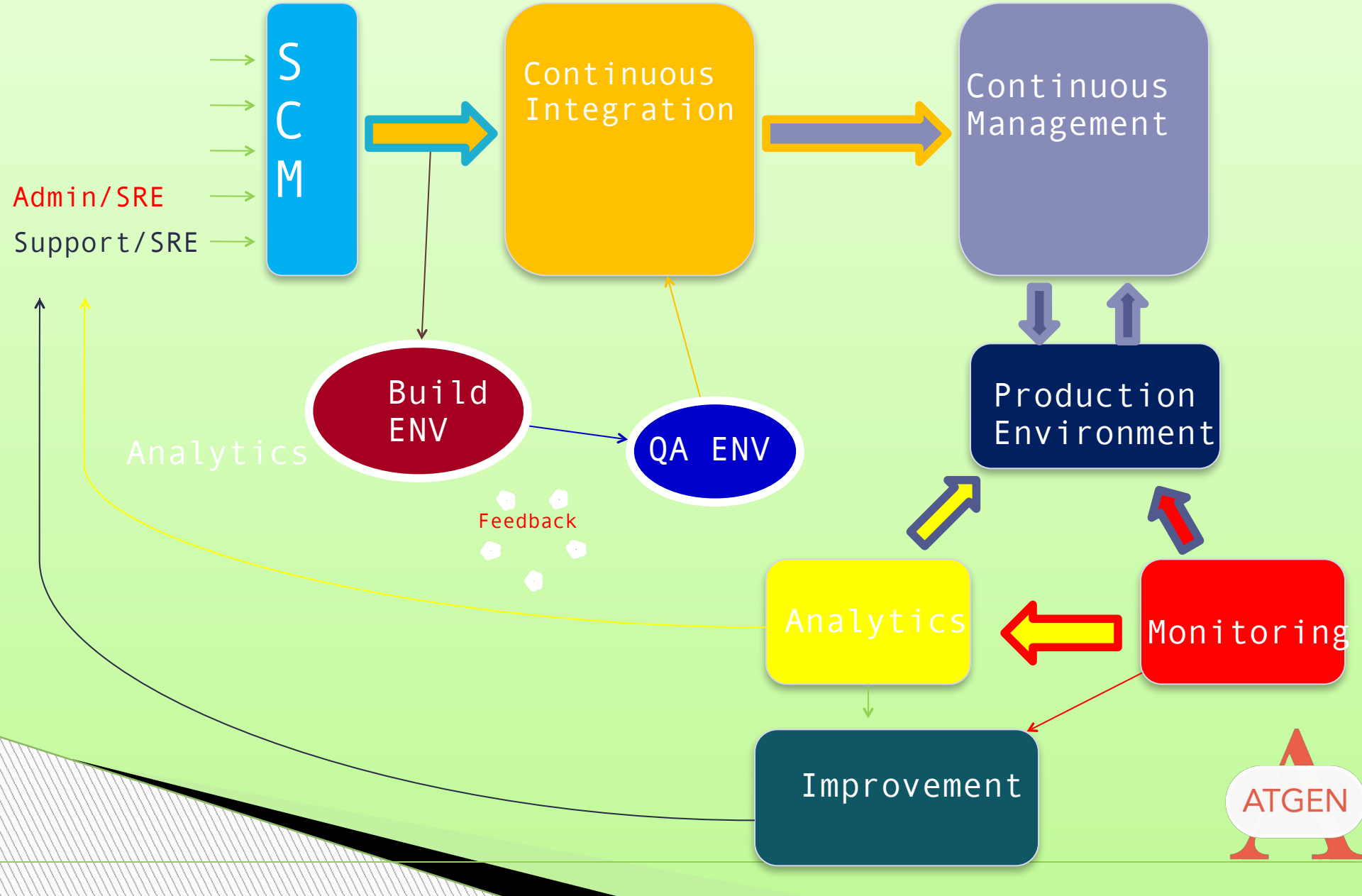


ATGEN

What is DevOps and SRE Role?



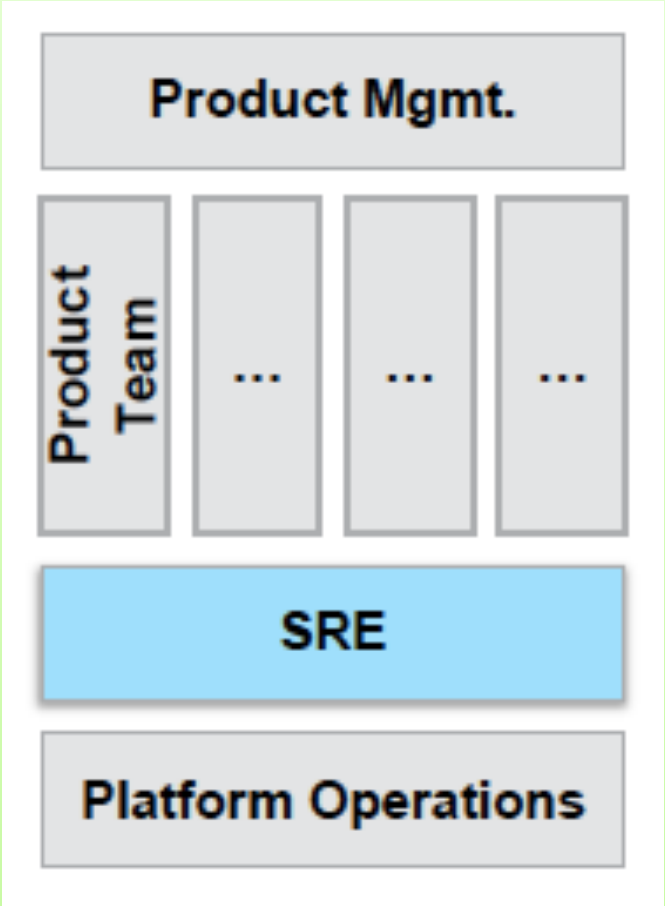
What is SRE?


S
C
M
Admin/SRE
Support/SRE
Continuous Integration
Continuous Management
Build ENV
QA ENV
Production Environment
Analytics
Feedback
Analytics
Monitoring
Improvement
ATGEN

Enabling SREs

Centralised SRE as a shared service

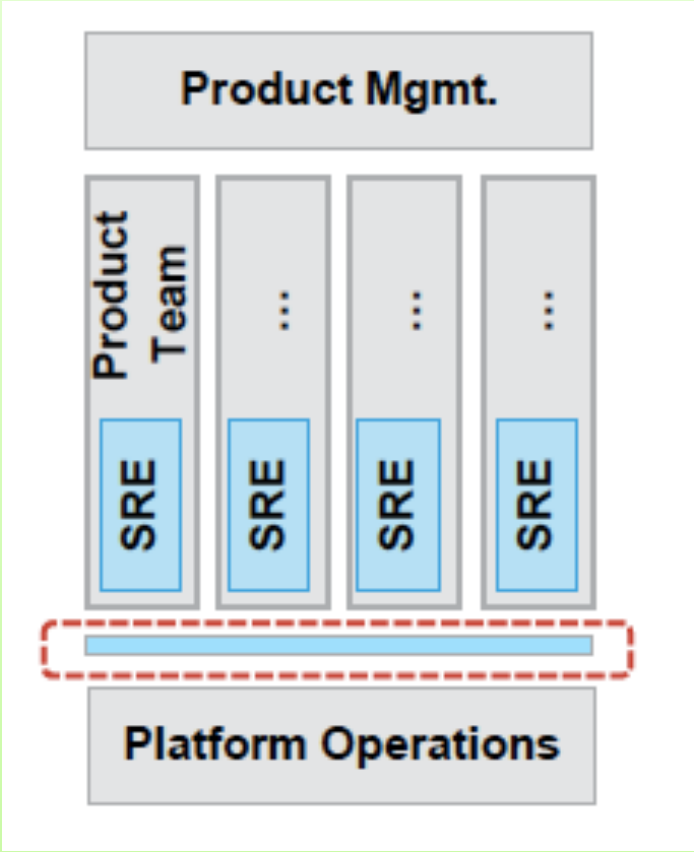
- Common backlog of organisation's reliability improvements – so rolled out evenly
- SREs are brought in as part of design, review, validate, go-lives
- Limited level of influence within team
- For Example, Facebook



ATGEN

Horizontal enablement across products

- Treated like a team member of the product team
- Regular contribution to the overall product reliability design & implementation
- For Example, Google, Bloomberg, Uber etc.



ATGEN

SRE - Key Metrics

- Mean Time to Restore (MTTR) service
- Lead time to release (or rollback)
- Improved monitoring to catch & detect issues earlier
- Establishing error budgets to enable budget-based risk management

SRE - Hiring SREs

- Ideally, multi-skilled individuals who have experience in writing software and maintaining software.
- Traditional Ops / Sys Admins who are willing to learn scripting / programming.
- App developers who are willing to step up for operational roles.
- Exposure to and interest in systems architecture.

SRE - Onboarding SREs

- SRE Bootcamps that consist of insight into the system architecture, software delivery process and operations.
- On-call rotations with senior SRE members.
- Participation in incidents response calls.

Budgeting for failures (Error Budgets)

SRE - Defining Service Levels

- Product Management with SREs define service levels for the systems as part of product design.
- Service levels allow us to:
  - Balance features being built with operational reliability.
  - Qualify and quantify the user experience.
  - Set user expectations for availability and performance.

SRE - Defining Service Levels

Service Level Agreement (SLA)

- A business contract the service provider + the customer
- Loss of service could be related to loss of business
- Typically, an SLA breach would mean some form of compensation to the client

Service Level Objective (SLO)

- A threshold beyond which an improvement of the service is required
- The point at which the users may consider opening up support ticket, the "pain threshold", e.g., YouTube buffering
- Driven by business requirements, not just current performance

Service Level Indicator (SLI)

- Quantitative measure of an attribute of the service, such as throughput, latency
- A directly measurable & observable by the users – not an internal metric (response time vs. CPU utilization)
- Could be a way to represent the user's experience

SRE - Measuring Reliability

$$\text{Availability} = \frac{\text{actual system uptime}}{\text{anticipated system uptime}}$$

- Easy to measure through systematic means (e.g., server uptime).
- Difficult to judge success for distributed systems.
- Does not really speak on behalf of the user's experience.

SRE - Measuring Reliability

$$\text{Availability} = \frac{\text{good interactions}}{\text{total interactions}}$$

- Measures the amount for real users for whom the service is working.
- Better for measuring distributed systems.

SRE - Budget for Unreliability

100% - Availability target = Budget of unreliability

or, the **Error Budget**

- Taking one step further, setup a budget that allows for taking risks.
- Method:
  - SRE and Product Management jointly define an availability target.
  - After that, an error budget is established.
  - Monitoring (via uptime) is set in place to measure performance + metrics relative to the objective set.
  - A feedback loop is created for maintaining / utilizing the budget.

SRE - Benefits of Error Budgets

- Shared responsibility for uptime.
- Common incentive for Product teams & SREs.
- Product teams can prioritise capabilities.
- Reliability goals become credible & achievable.

SRE - SLO & Error Budget

SLO

99.9%

Error Budget

43.2 Min / Month

$$\frac{99.9\% \uparrow}{\text{month}} = \frac{0.1\% \downarrow}{\text{month}} = 0.001 \times \frac{30\ \text{day}}{\text{month}} \times \frac{24\ \text{hour}}{\text{day}} \times \frac{60\ \text{min}}{\text{hour}} = \frac{43.2\ \text{min}}{\text{month}}$$

ATGEN

SRE - Unavailability Window

Availability Target	Allowed Unavailability Window	
	per year	*per 30 days*
95 %	18.25 days	1.5 days
99 %	3.65 days	7.2 hours
99.9 %	8.76 hours	43.2 minutes
99.99 %	52.6 minutes	4.32 minutes
99.999 %	5.26 minutes	25.9 seconds

ATGEN

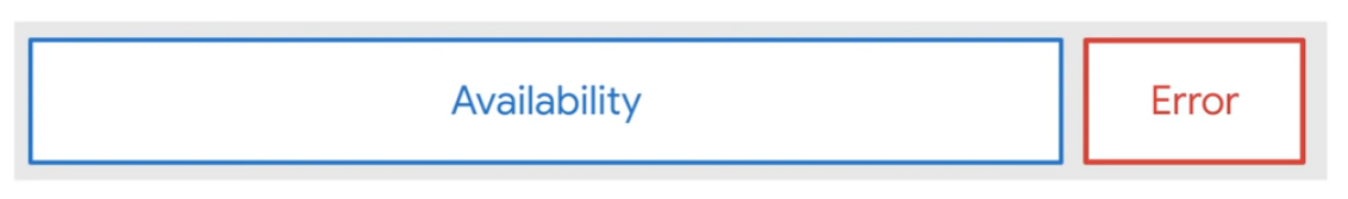
SRE - Exhausting an Error Budget

- Question: what can you do if you exhaust the error budget?

- Possibly:
  - No new feature launches (for a period of time)
  - Change the velocity of updates
  - Prioritise engineering efforts to reliability

As long as there is remaining error budget (in a rolling timeframe), new features can be deployed.

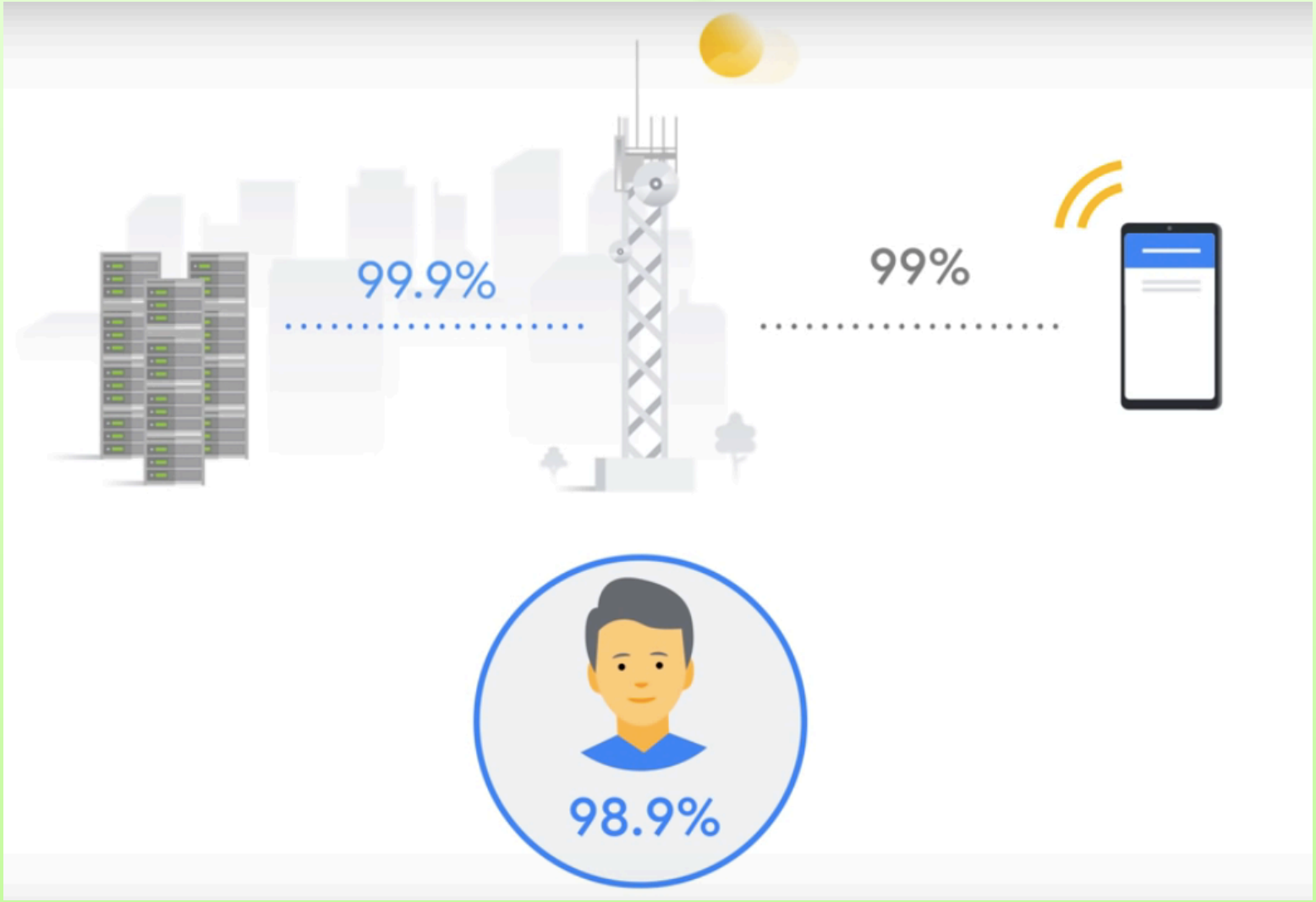
SRE - Error Budget Estimation



- Total Error should constitute errors at every level in your service stack.For Example,

ATGEN

SRE - Error Budget Estimation



ATGEN

Eliminating Toil

SRE - What is toil?

toil |tɔɪl| *n.*

1. Repetitive, manual work that is associated with some operation. Scales linearly with the size of a service and doesn't provide lasting value.

SRE - Examples of Toil

- Manually reviewing monitoring and logging systems everyday to make sure your system is till running.
- Deploying software to many machines via scp.
- Turning up a service in a new availability-zone or region requires copy-pasting configuration, manually migrating data.
- Common tasks are driven by tickets: users request “thing 1” so they file a ticket asking an operator to run some script do_thing_1.sh

SRE - Toil is Bad

- Systems that are very toilsome are expensive. The require scaling headcount than the service.
- Toil accumulates.
- Error Prone.
- Toil isn’t exciting!

SRE - Measure Toil

- Don’t just rely on intuition, humans are really bad at estimating how much time they spend on certain tasks.
- Look for opportunity to reduce toil.

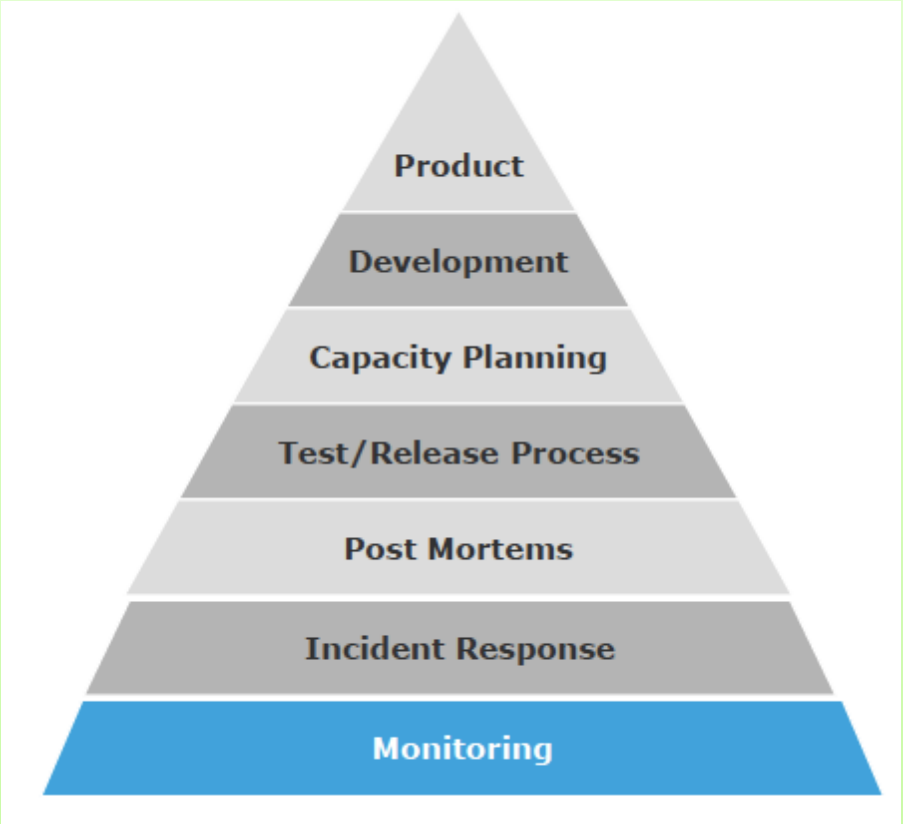
SRE - Toil Reduction Patterns

- Use Third Party/Open Source where possible.
- Standards for rollouts.
- Standard place to put ‘automations’.
- Don’t accept toil in first place.

Measuring the Systems (Monitoring & Alerting)

SRE Hierarchy - Monitoring

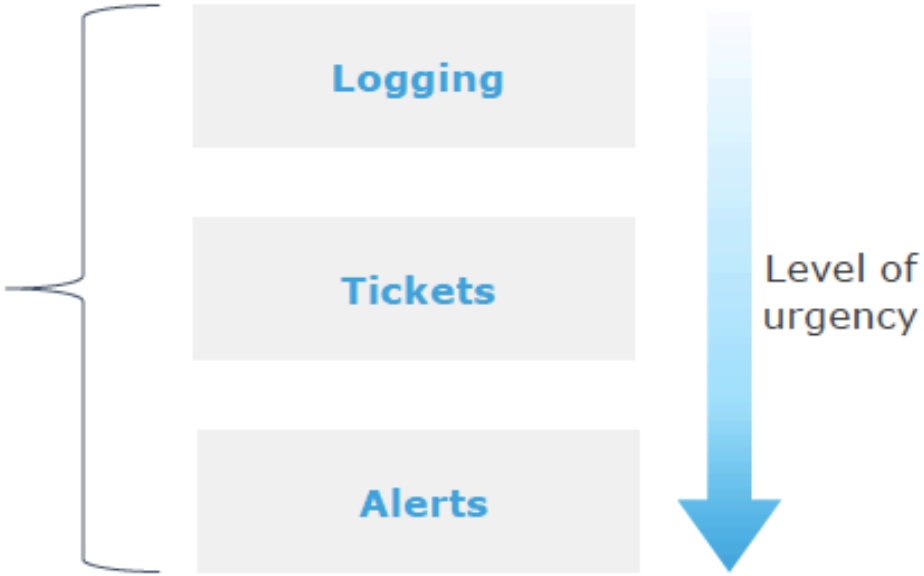
- Monitoring helps with identifying what needs urgent attention, and helps prioritization:
  - Urgency
  - Trends
  - Planning
  - Improvements



ATGEN

SRE - Good Monitoring

A carefully crafted **monitoring system** should be able to distinguish urgent requests from important ones, and only alert when required



ATGEN

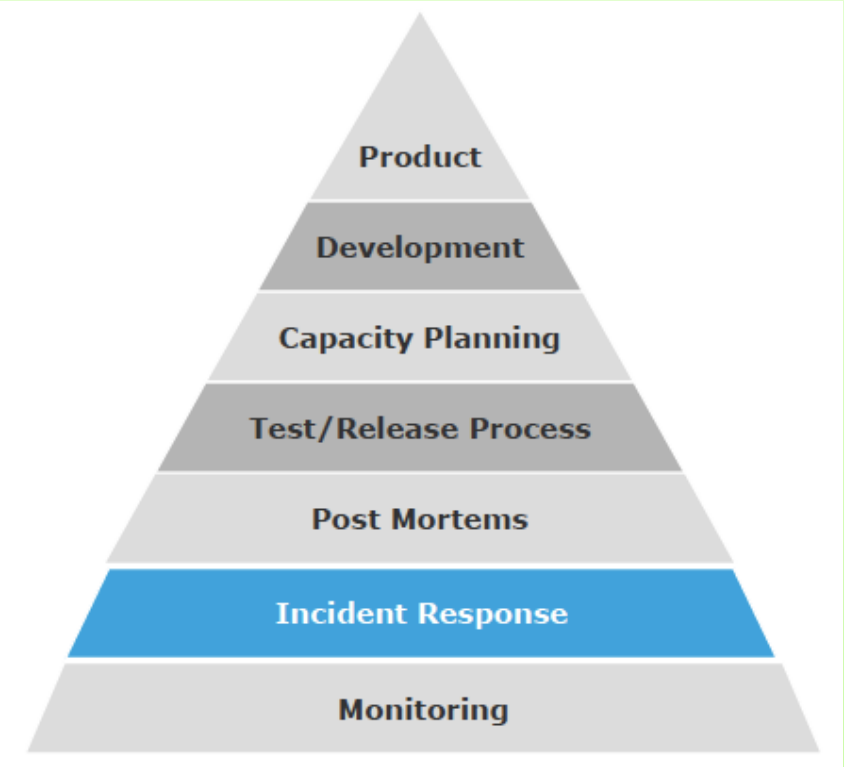
SRE - Alert Design Consideration

- An alert should only be paged for urgent requests that need immediate attention.
- Common Issues:
  - Not all alerts are helpful
  - Alerts is not maintained regularly to keep up with product functionality being rolled out
- Good practices:
  - Measure the system along with all of the parts, but alert only on problems will cause or is causing end-user pain. This helps to reduce “alert fatigue”
  - Alerts should be actionable (consider logging for informational items)

Managing Incidents (Incident Response & On-Calls)

SRE - Incident Response

- Effective incident response process increases user trust.
- Users don’t know always expect a service to be perfect.
- However, they expect their service provider to be transparent, fast to resolve issues, and learns quickly from mistakes.
- Incident Response helps to create that sense of trust when there is a crisis.



ATGEN

SRE - Structured Incident Response

- People do not typically work well in situations of emergencies, so having a structured incident response process is essential.

SRE - Structured Incident Response

1

Have sufficient monitoring

- Service dashboards indicating throughputs, loads, latency, etc.
- Typically, an SLA breach would mean some form of compensation to the client

2

Good alerting & communication process

- Notifying on-call member(s) and having an automated escalation process if primary on-call is not reachable
- Notifying end-users before or as soon as there is degraded experience

3

Playbook-style plans & tools

- An easily accessible step-by-step checklist of actions to do when an incident occurs
- Tools available for system introspection

SRE - Post Mortems

- A post-mortem conducted after an incident helps to build a culture of self-improvement.

- Goals of writing a post-mortem:
  - All contributing causes are understood and documented
  - Action items are put in place to avoid repeating the same issue again

- Good habits:
  - Post-mortems conducted & closed within 3-5 days after an incident has occurred
  - Document in details errors, logs, steps taken and decisions made that led to the incident/outage
  - Document fixes if an SLO is breached, even if the SLA is not impacted

SRE - blameless post-mortem

- Blameless culture enables faster experimentation without fear. Because failures are part of the process of experimentation.
- “Human errors” are system problems.
- It’s better to fix systems & processes to better support people in making good choices.
- If the culture of finger-point prevails, there is no incentive for team members to bring forth issues in fear of punishment.

SRE - Conduct Five Why's?

- How can you provide better information to make decisions?
- Was the information misleading?
- Can this information be fixed in an automated way?
- Can the activity be automated so it requires minimal human intervention?
- Could a new hire have made the same mistake the same way?

SRE - Better On-Calls

- Create a work environments that have a balance between planned/project-driven work and interrupt-driven work.

Project-driven work	Interrupt-driven work
Planned work to reduce toil	Supporting incidents

SRE - Good practices On-Calls

- Have a single person be in charge of on-call
- On-call person makes sure interrupt-based work is completed without impacting others who are not on-call
- Create a rotation schedule to ensure relief
- No more than 2 key incidents per on-call engineer per shift(or staff appropriately)

Enablers (Technology components)

SRE - Technology capabilities

Cloud

Cloud architecture is a direct response to the need for agility.

Containers

Infrastructure manages services and software deployments on IT resources where components are **replaced** rather than **changed**.

Microservices

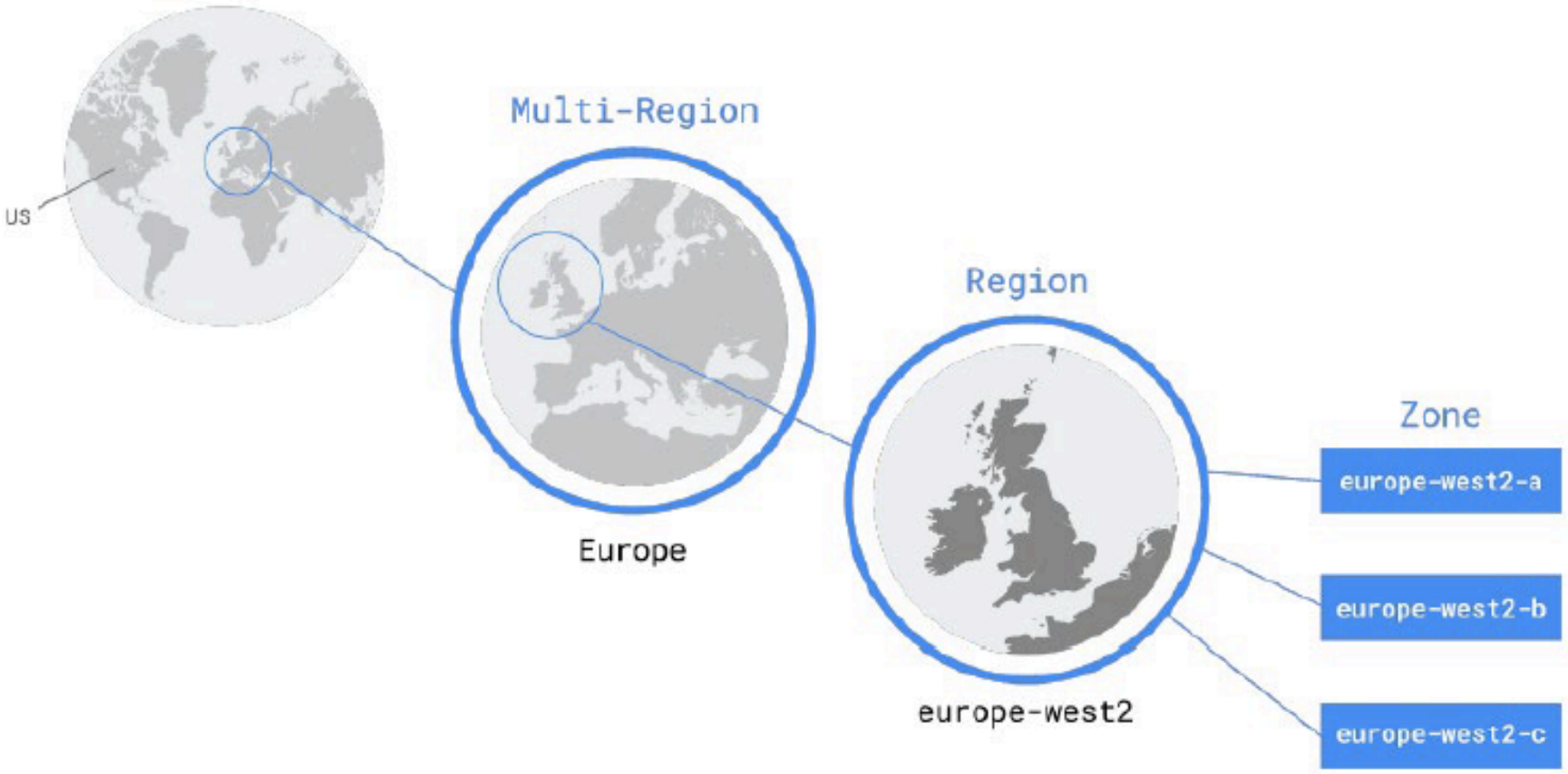
Microservices provide a foundation to support automation, rapid deployment, and DevOps infrastructure.

ATGEN

SRE - Infrastructure Design

- Version control your infrastructure design and code.
- Offload as much work to cloud provider.
- Design for reliability by having the application + data situated in more than one datacenter.
- Design your infrastructure architecture to support deployments.

SRE - Multiple Datacenters



ATGEN

SRE - Application Design

- Re-think how your features can be re-architected to support for scalability (hence increasing reliability).
- De-couple / modular architecture: such as Microservices.
- Distribute your applications across multiple geographies.
- Consistent build, deploy process.

SRE - The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

ATGEN

SRE - The Twelve Factors

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

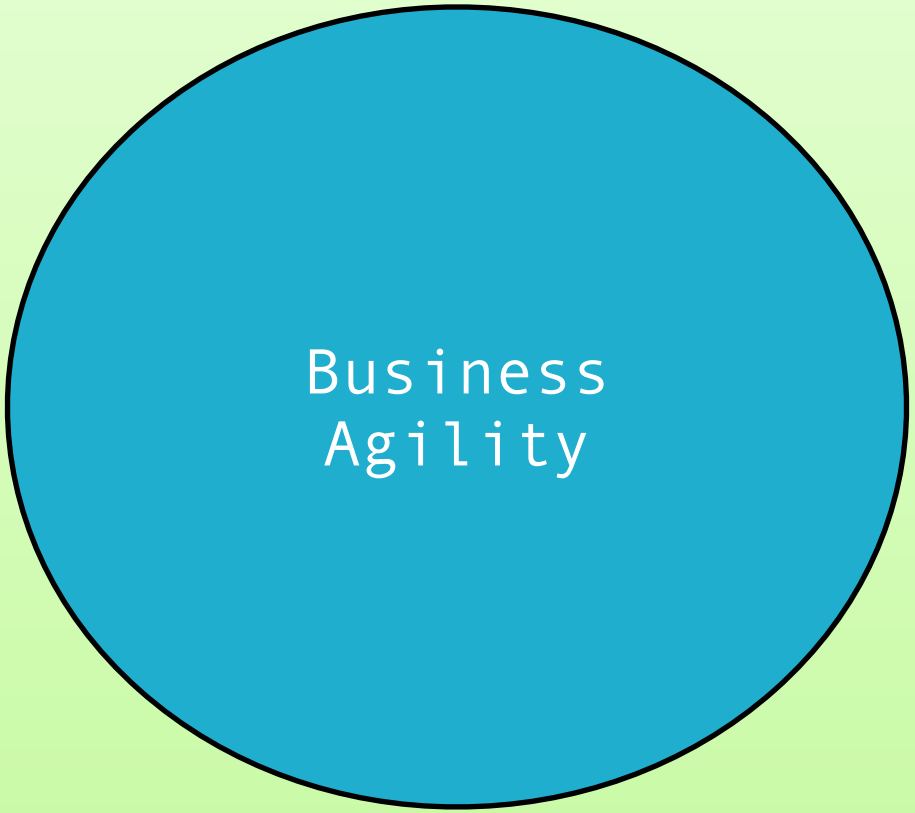
XII. Admin processes
Run admin/management tasks as one-off processes

ATGEN

Session: 2

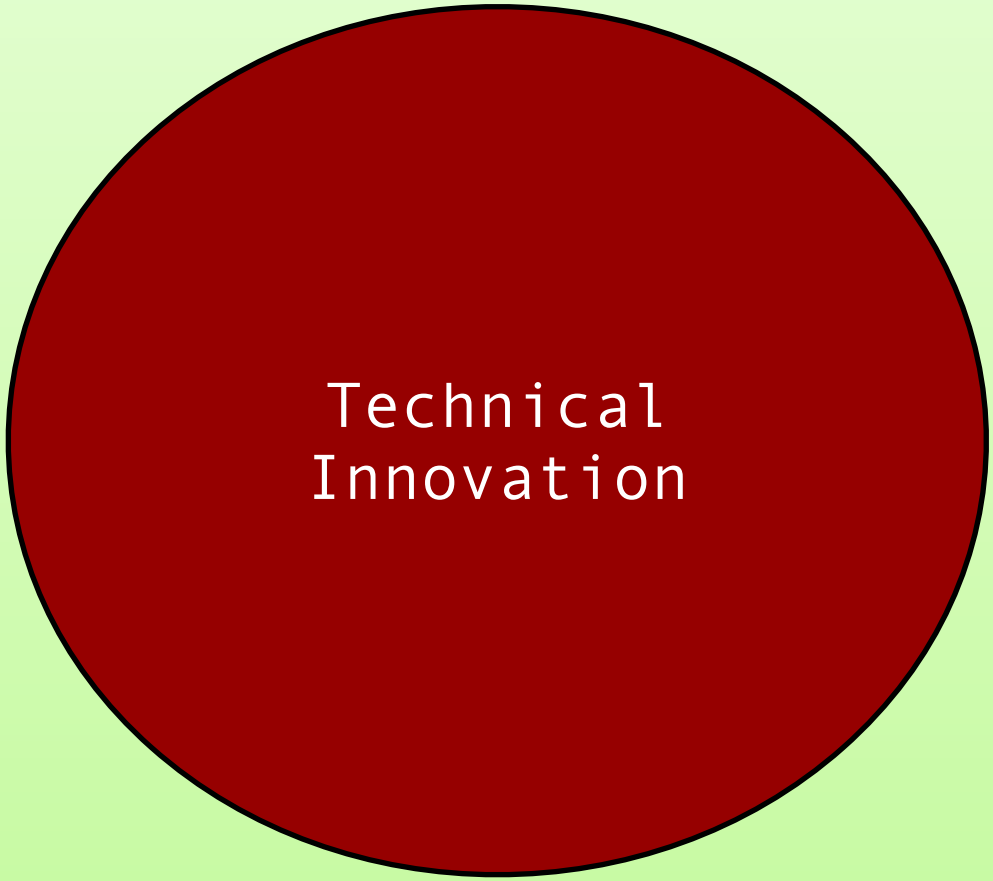
Why SRE?

SRE - Business Agility

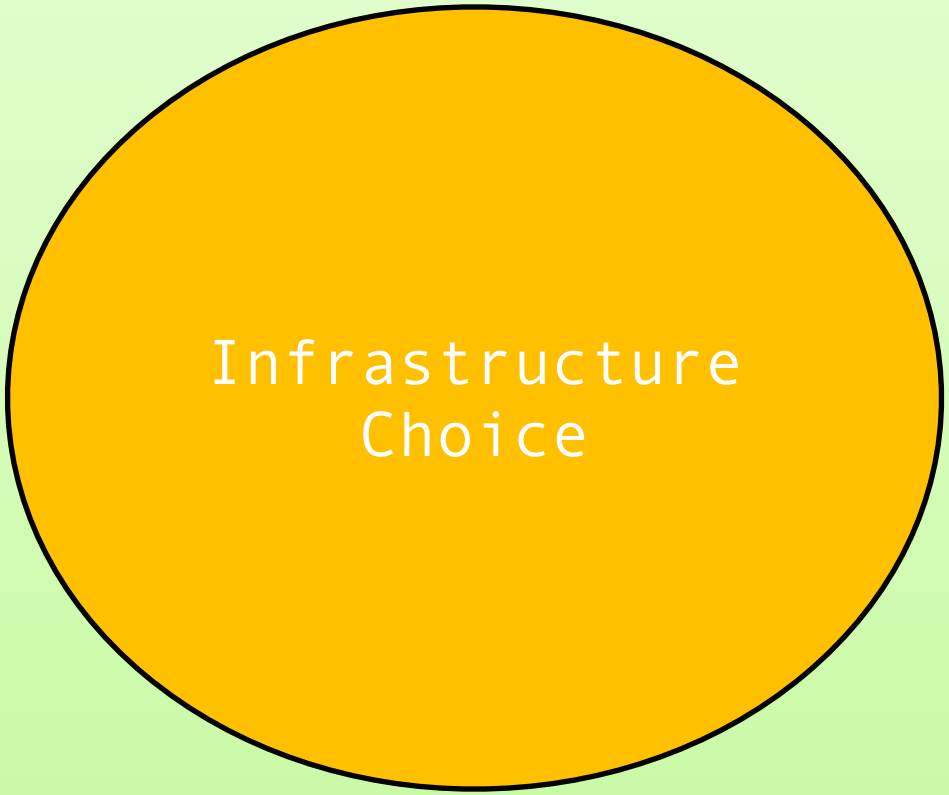


ATGEN

SRE - Technical Innovation

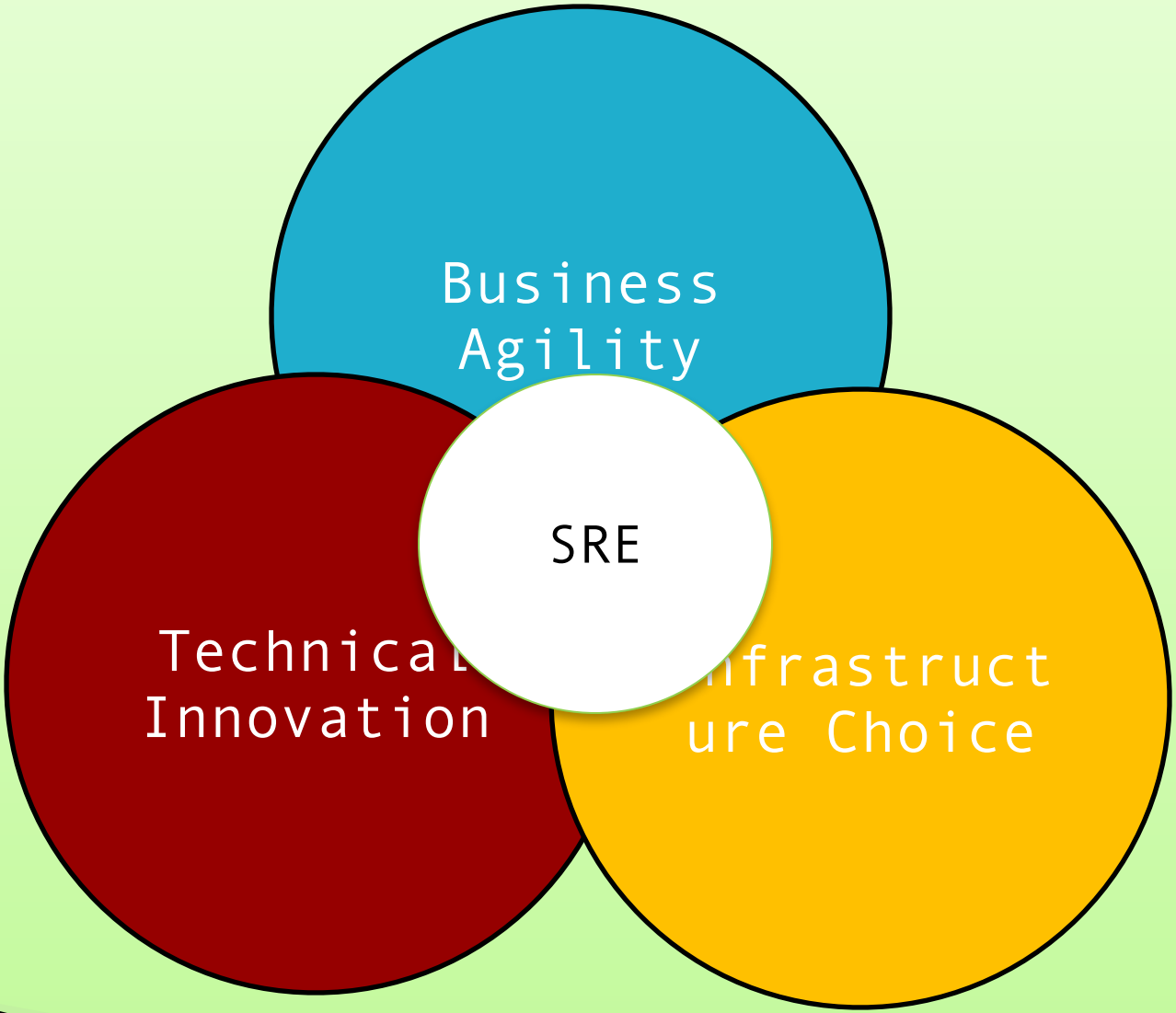


SRE - Infra Choice



ATGEN

SRE


Business
Agility
SRE
Technical
Innovation
nfrastruct
ure Choice

ATGEN

SRE – Culture is Yoghurt

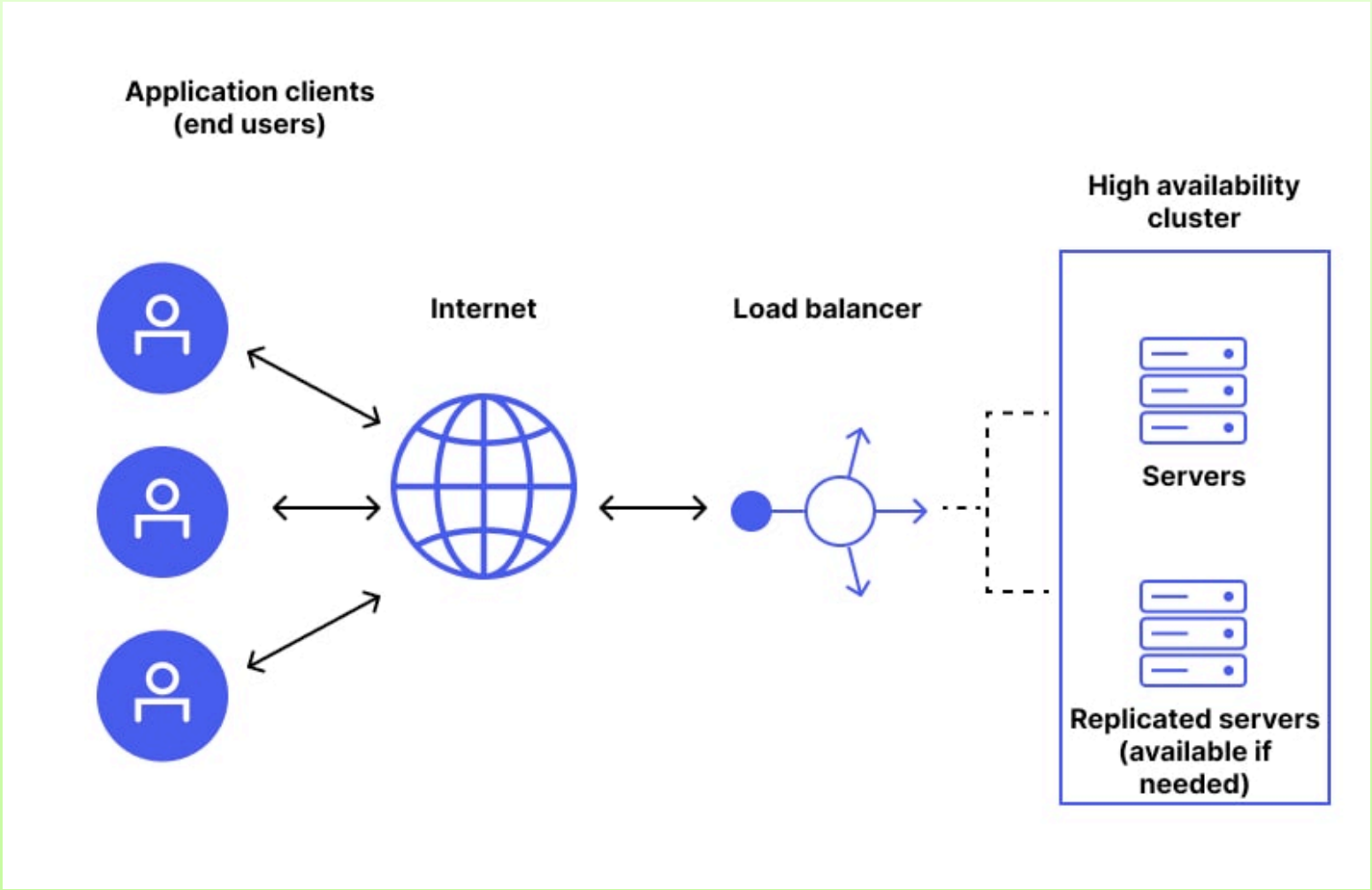
- Culture is an output, not an input
- Common organization with a nearby boss
- Shared metrics foundation of collaboration
- High-level sponsorship

ATGEN

The Essence of SRE

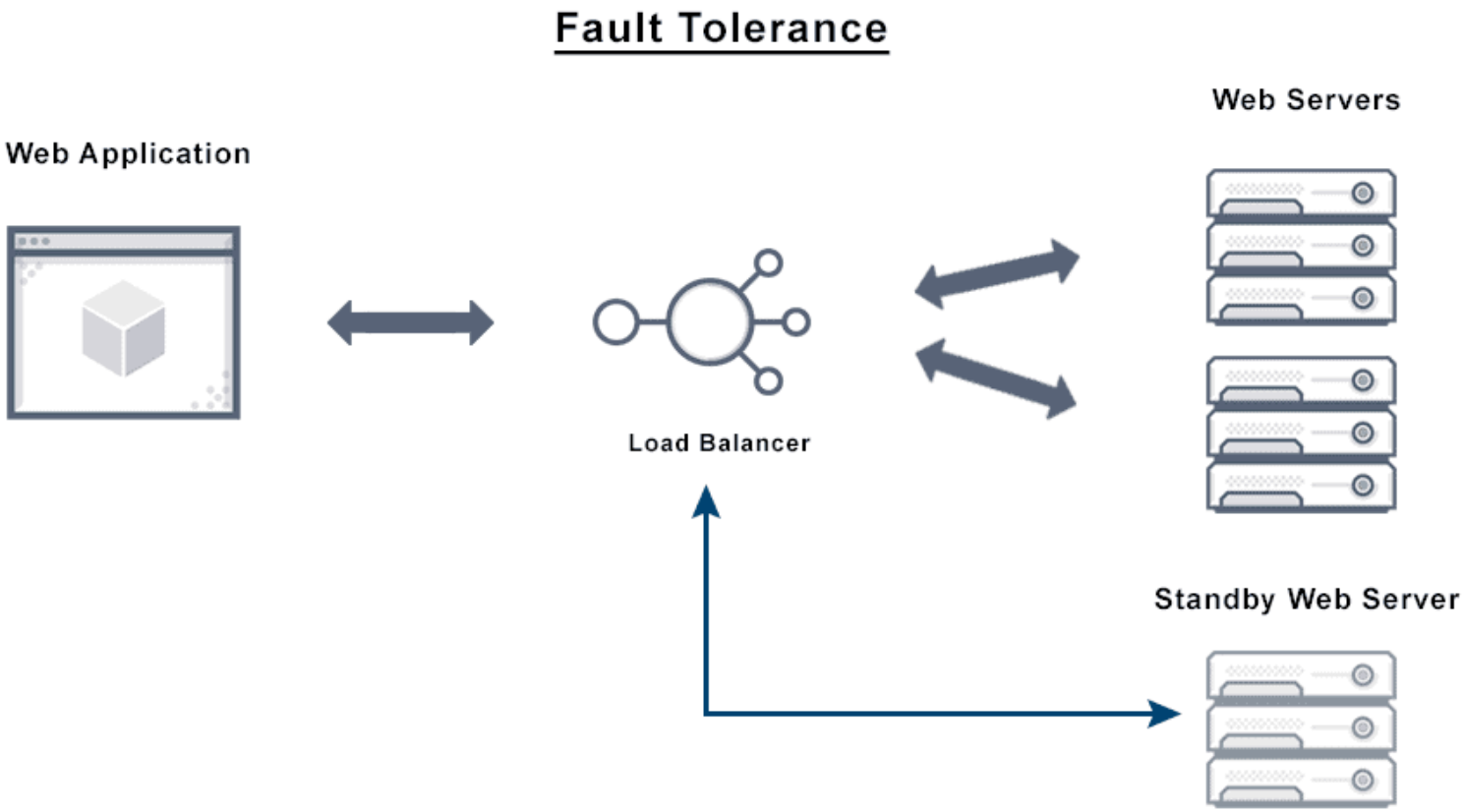
- Efficiency
- Predictability
- Reproducibility
- Maintainability
- Reliable Systems
- Better Customer Satisfaction
- Reduce Cost in Long Term

SRE - High Availability



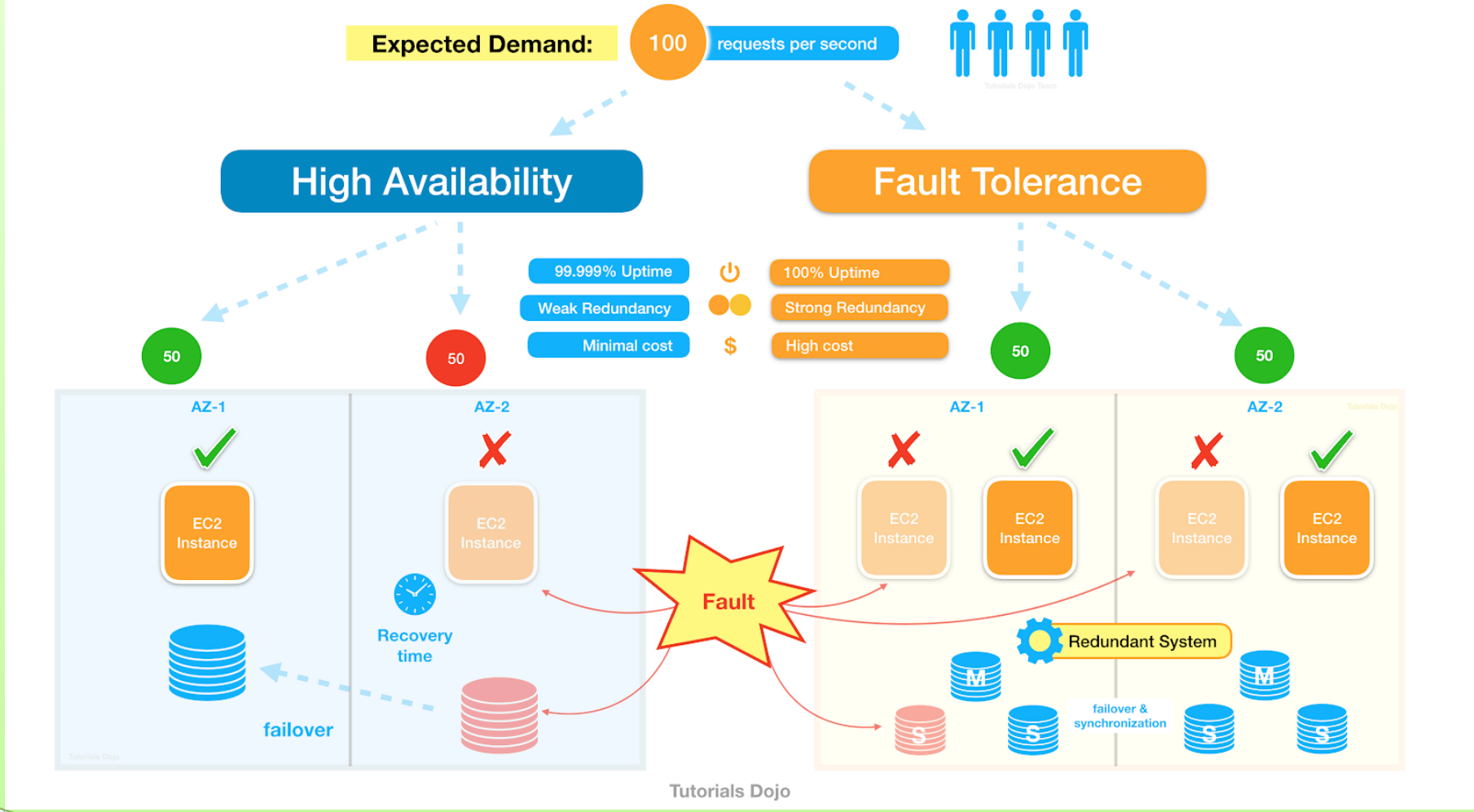
Atgen

SRE - Fault Tolerance



Atgen

SRE - HA vs FT



Atgen

Session: 3

SRE - 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.

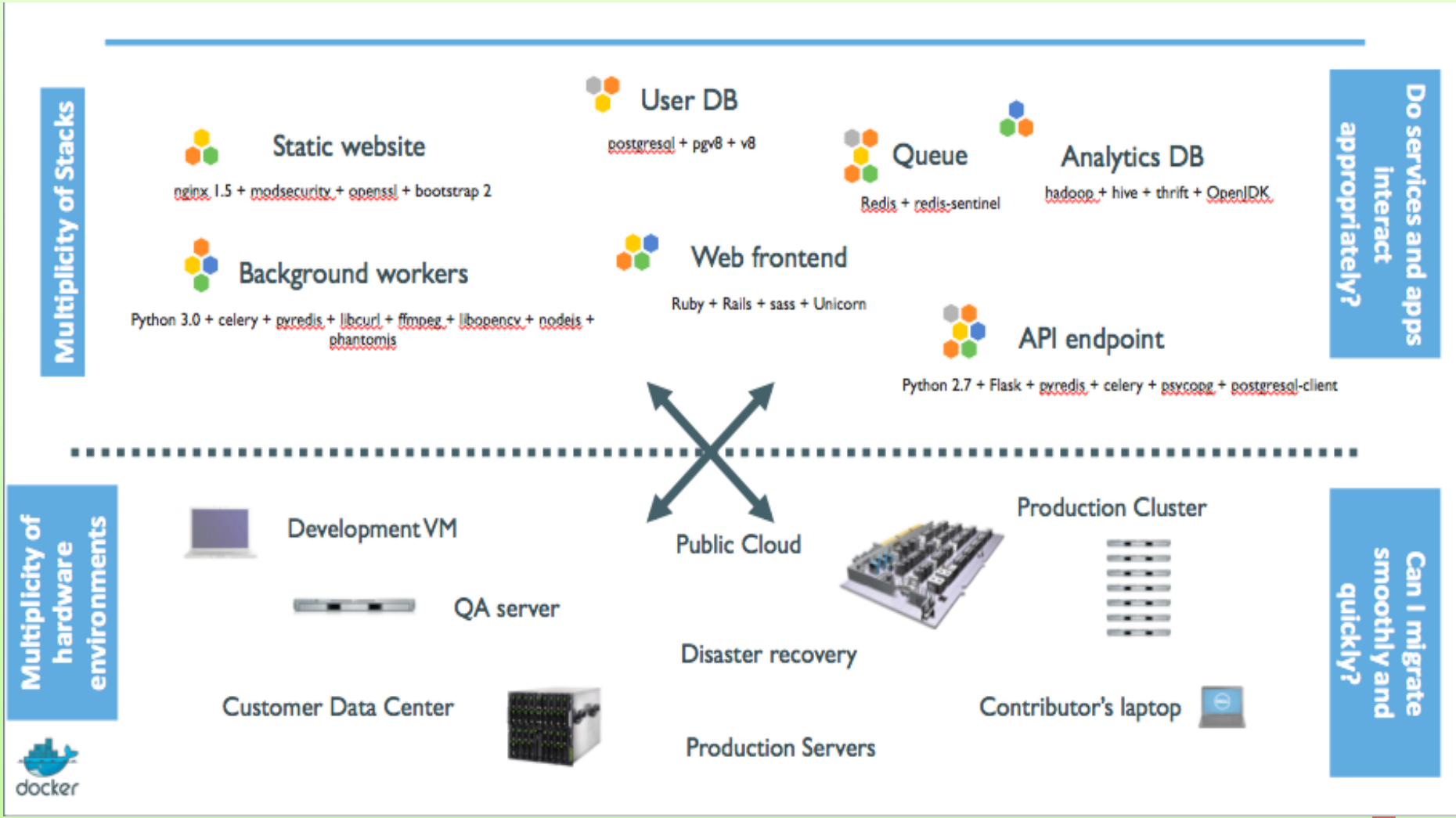
Docker becomes a platform

- 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

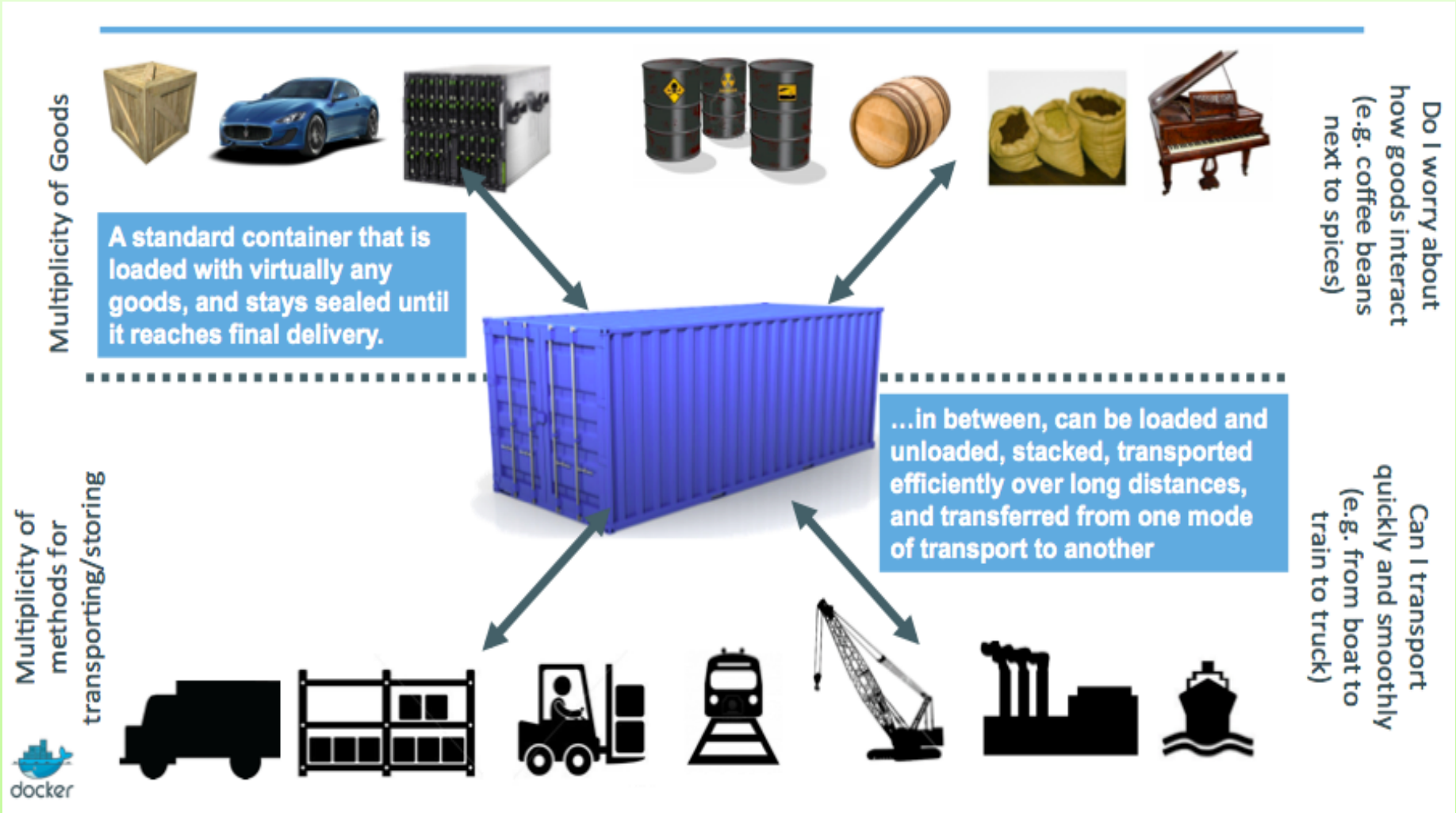
Matrix Checks

	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

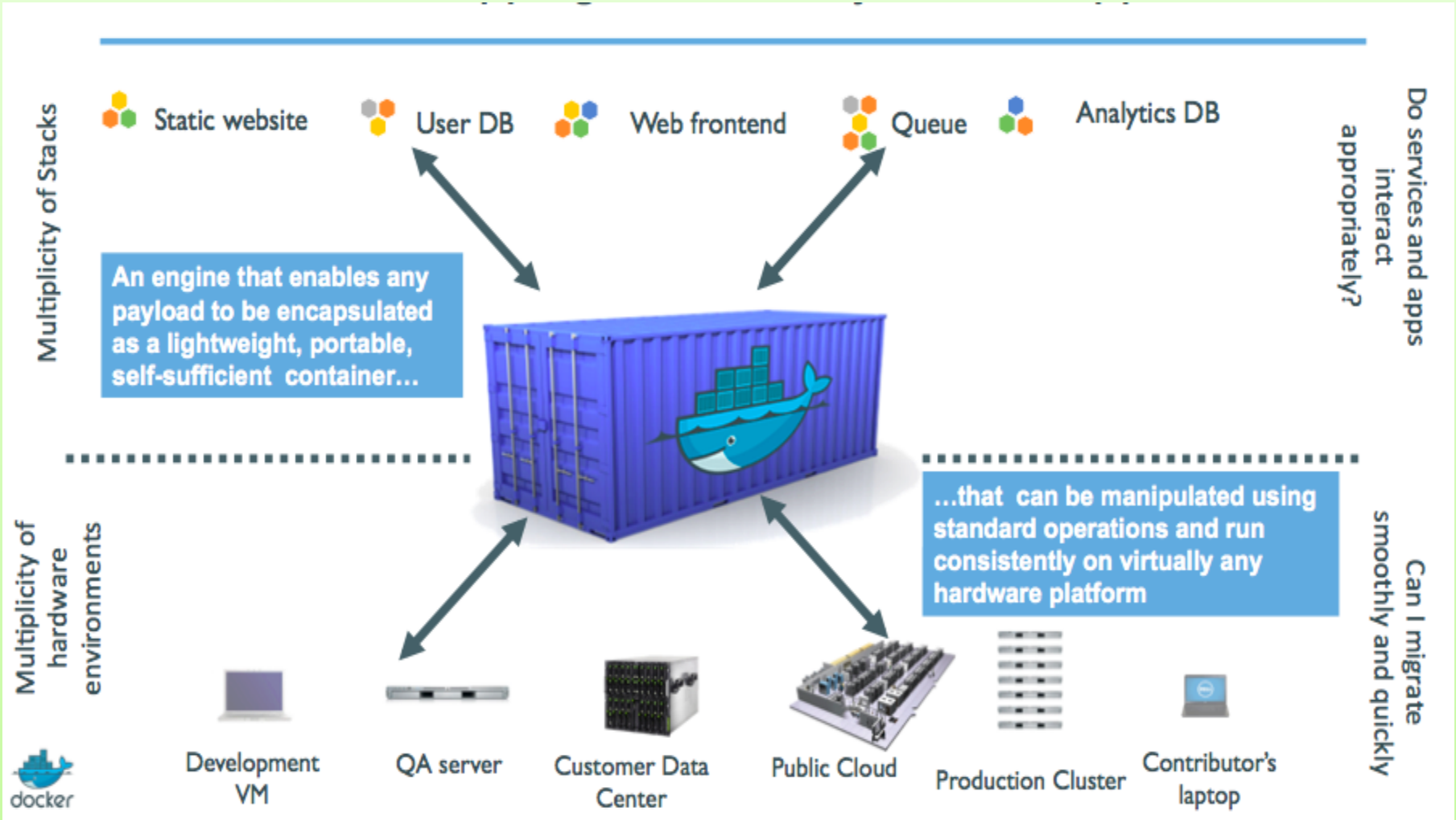
ATGEN

Intermodal Shipping Containers

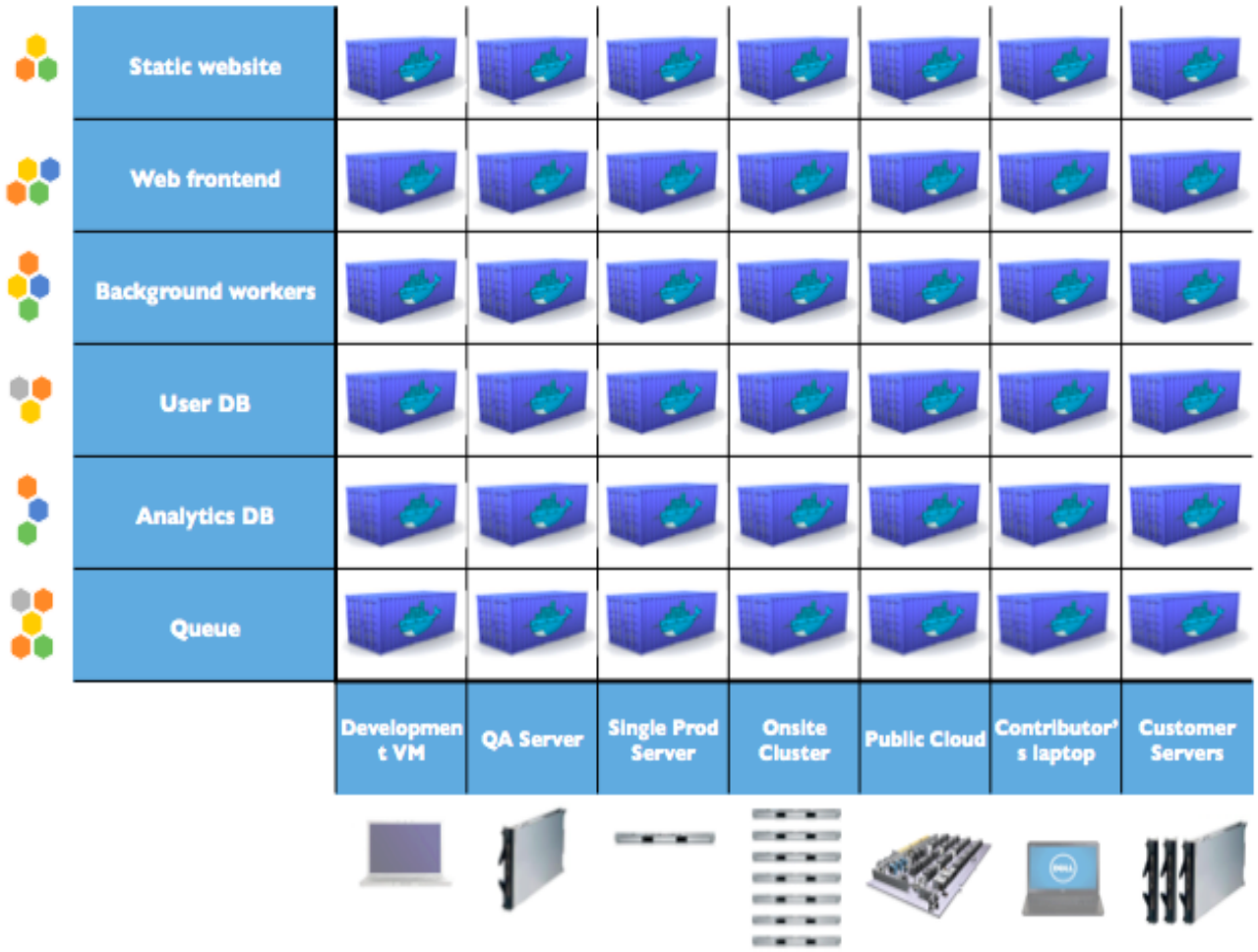


ATGEN

Shipping Container for Applications



Eliminate the Matrix



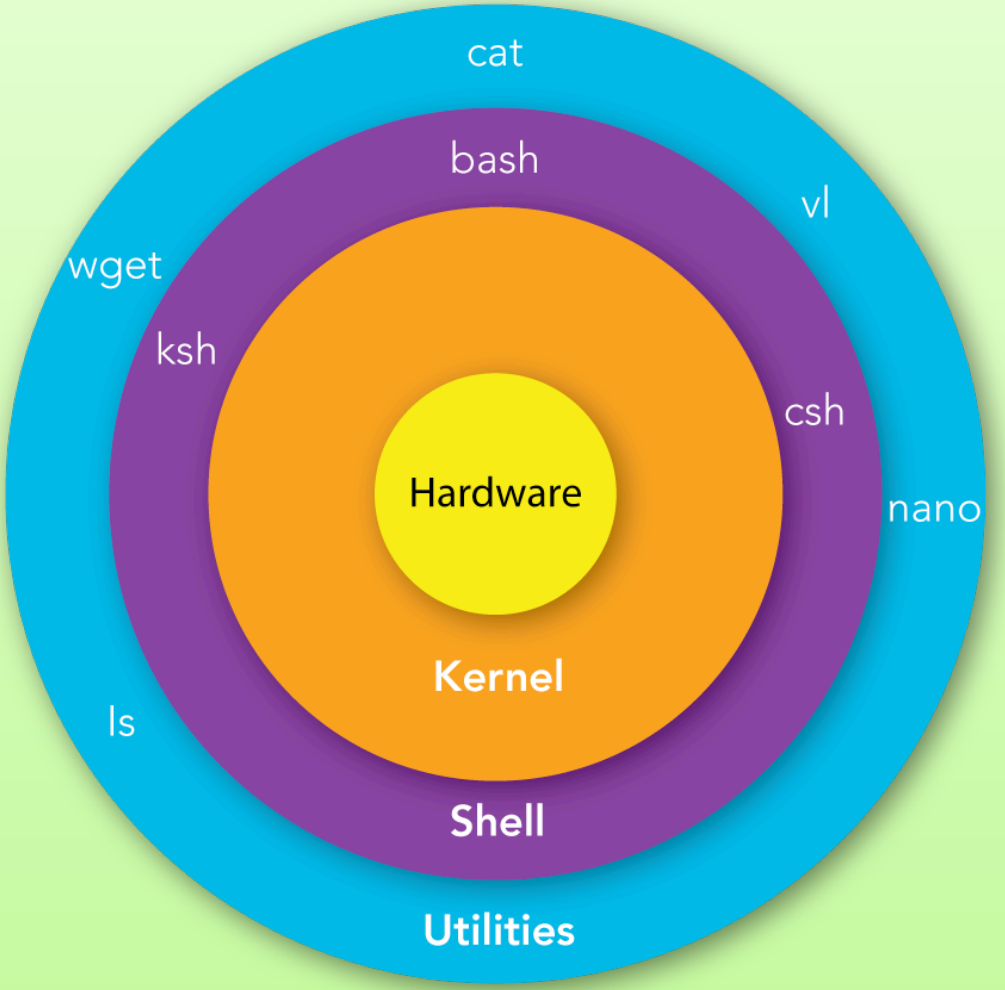
docker

ATGEN

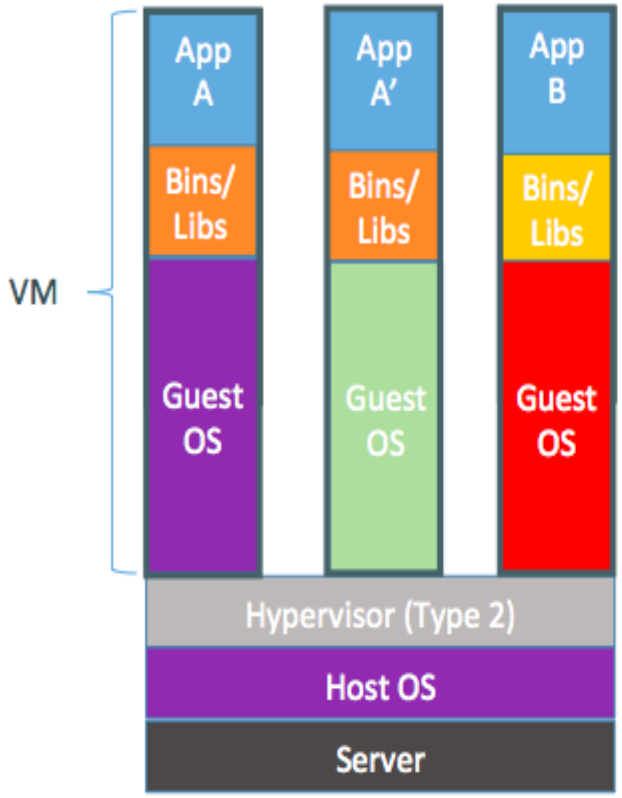
Results

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

Operating System Layers

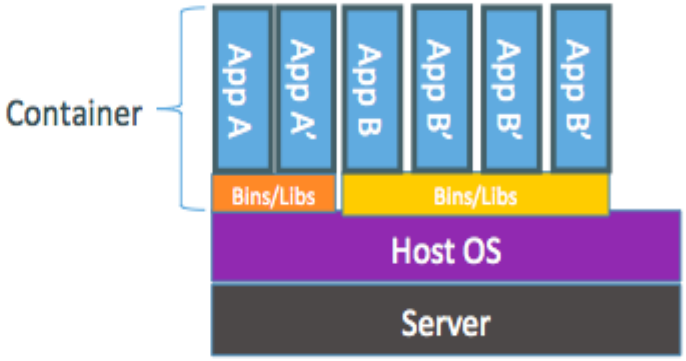


Containers = cheaper than VMs



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

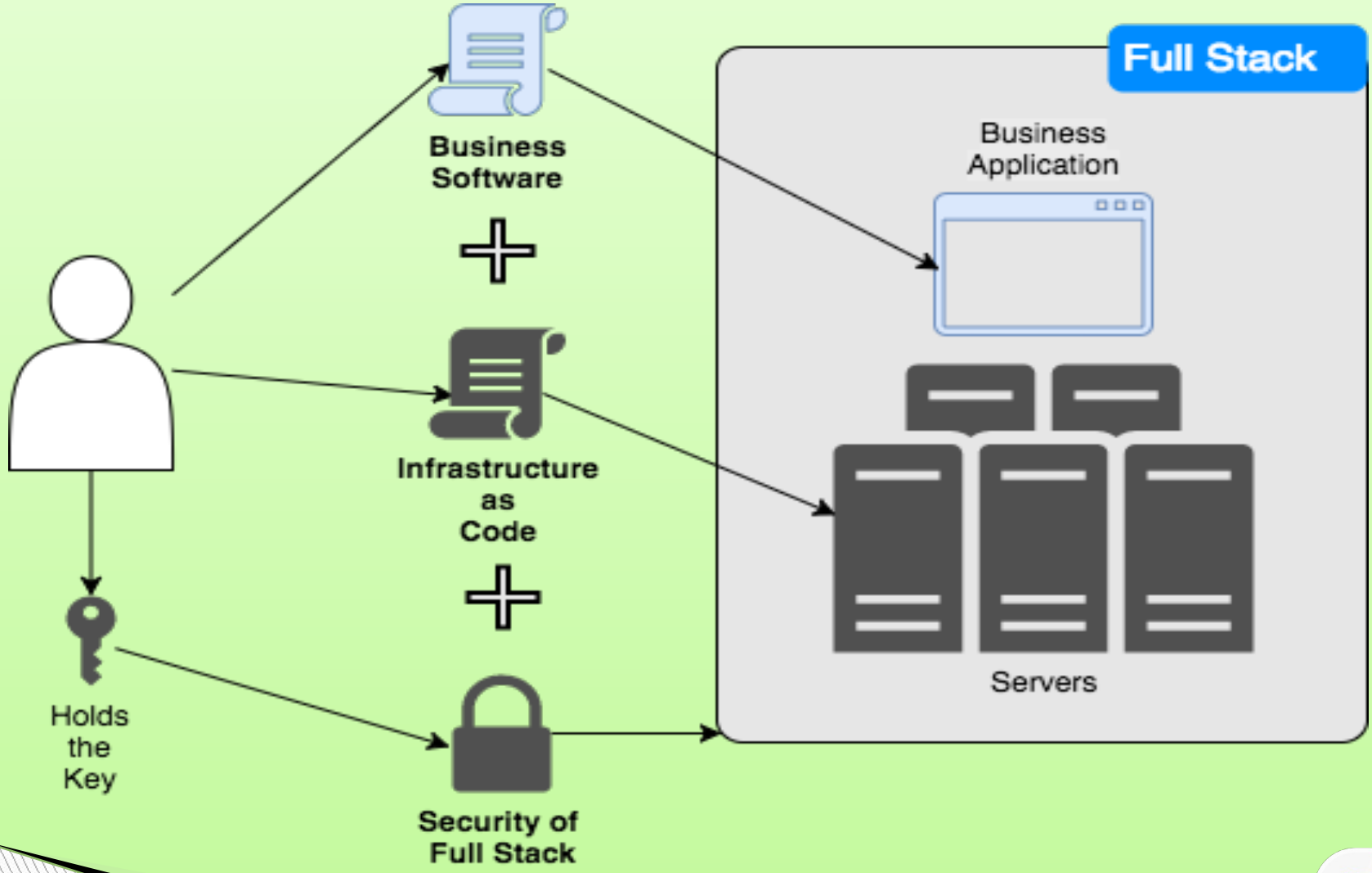
ATGEN

Session: 4

SRE - IaC

ATGEN

What is Business Service?



ATGEN

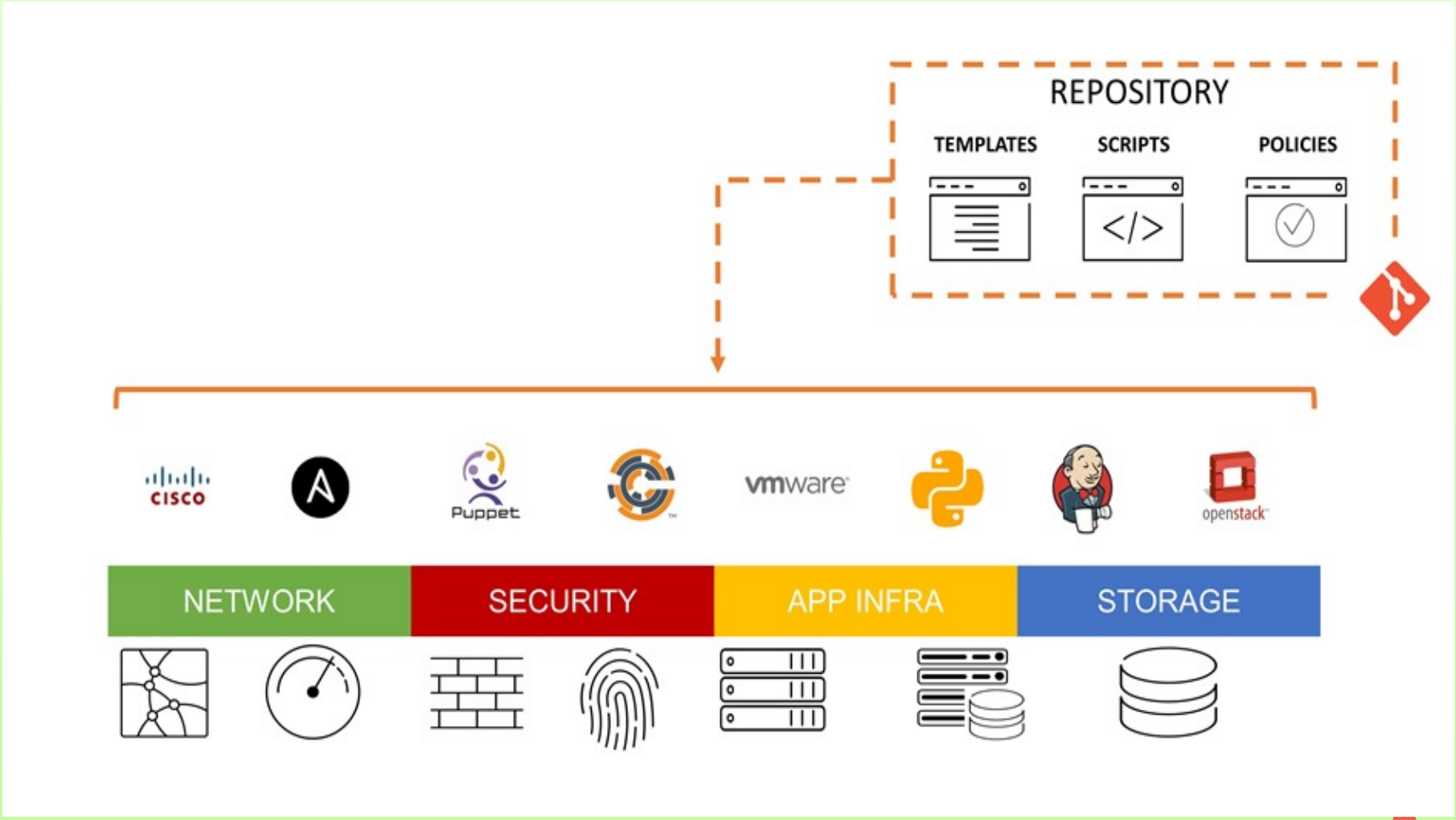
What is IaC?

- Infrastructure as code, also referred to as IaC, is a type of IT setup wherein developers or operations teams automatically manage and provision the technology stack for an application through software, rather than using a manual process to configure discrete hardware devices and operating systems.
- Infrastructure as code is sometimes referred to as programmable or software-defined infrastructure.
- The concept of infrastructure as code is similar to programming scripts, which are used to automate IT processes. However, scripts are primarily used to automate a series of static steps that must be repeated numerous times across multiple servers.

What is IaC?

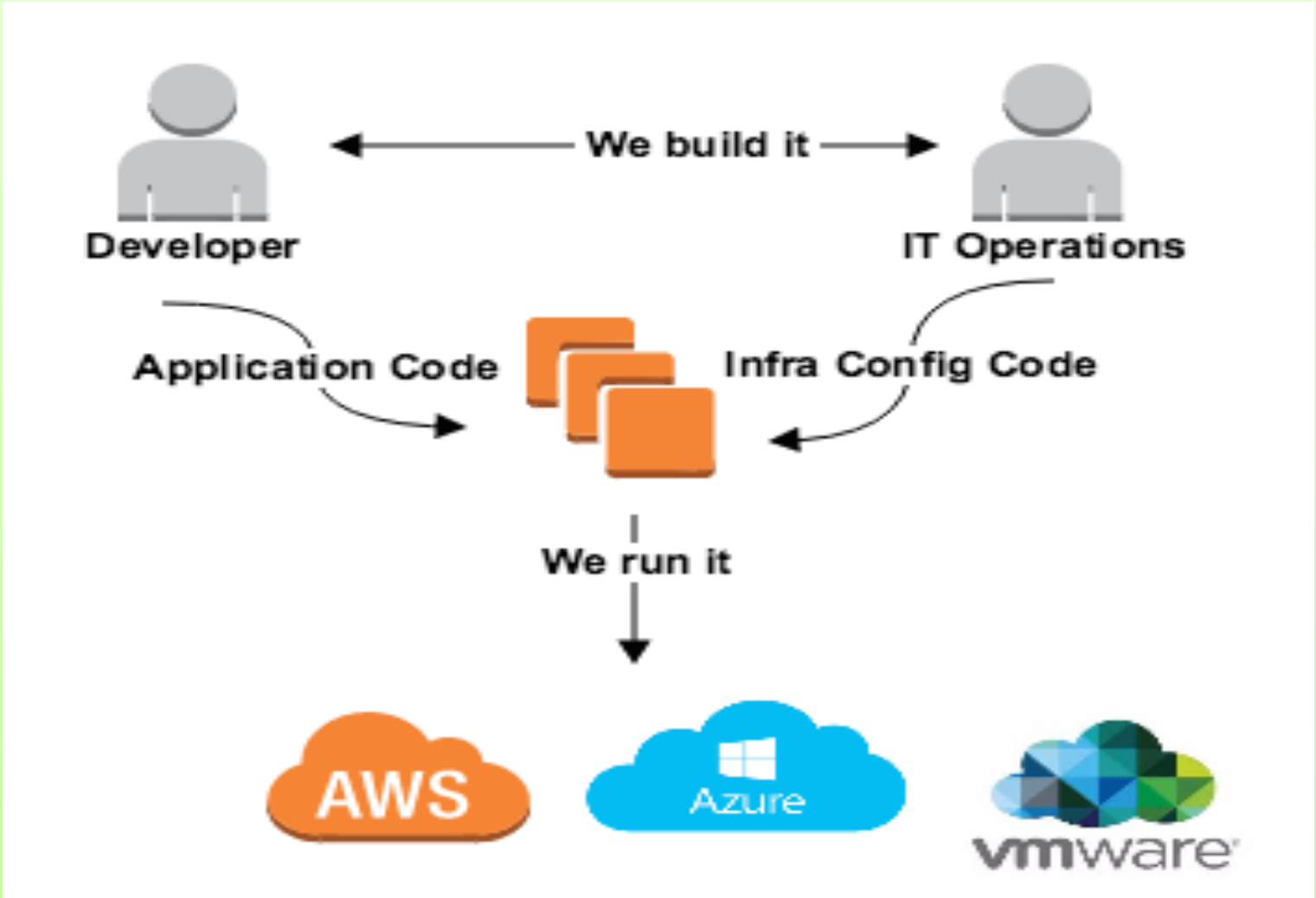
- Infrastructure as code uses higher-level or descriptive language to code more versatile and adaptive provisioning and deployment processes. For example, infrastructure-as-code capabilities included with Terraform, an IT management and configuration tool, can install WebServer, verify that WebServer is running properly, create a user account and password.
- The code-based infrastructure automation process closely resembles software design practices in which developers carefully control code versions, test iterations, and limit deployment until the software is proven and approved for production.

What is IaC?



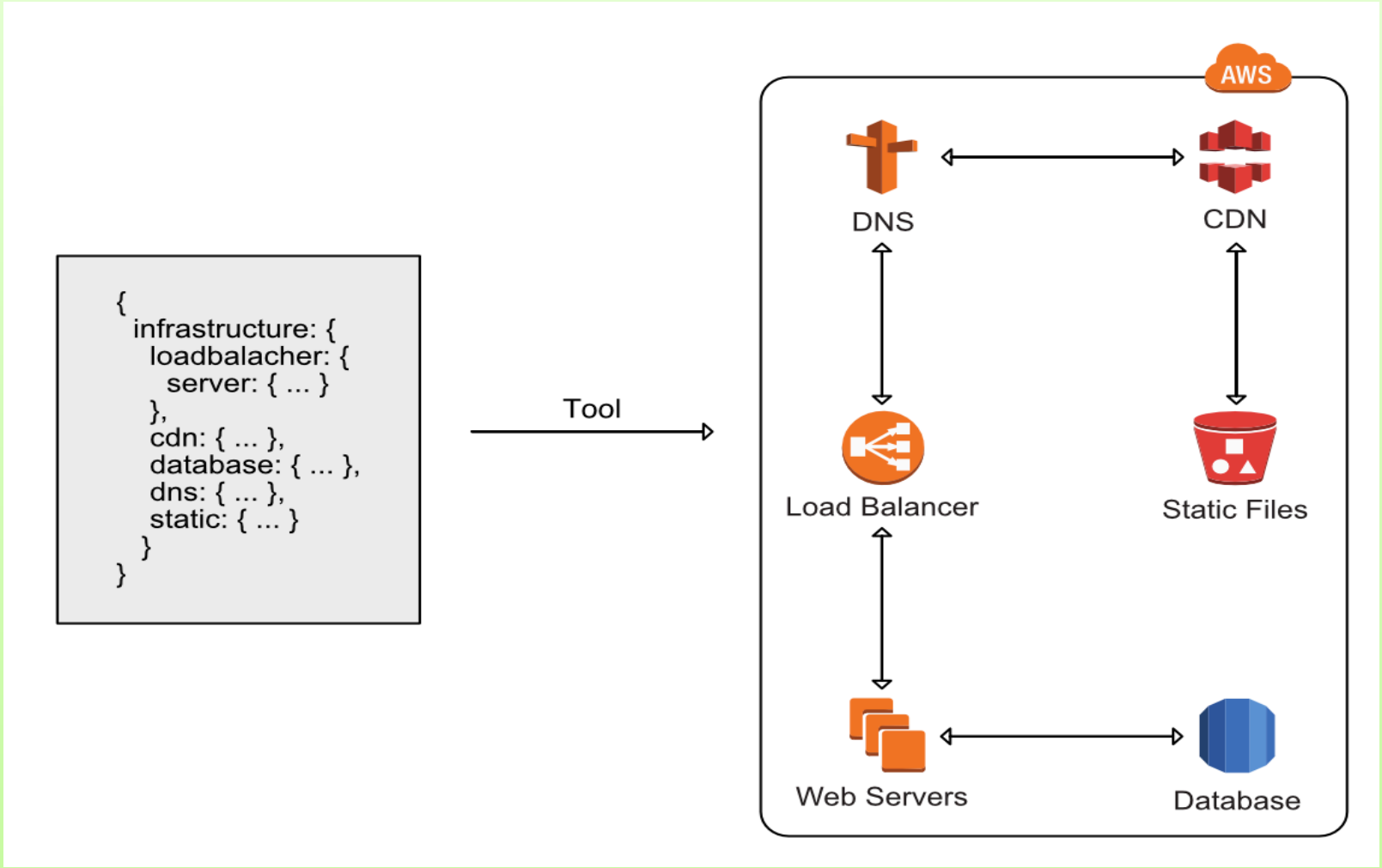
ATGEN

Where we stand?



ATGEN

IaC – Example



ATGEN

Benefits of IaC

- Software developers can use code to provision and deploy servers and applications, rather than rely on system administrators in a DevOps environment.
- With the infrastructure setup written as code, it can go through the same version control, automated testing, and other steps of a continuous integration and continuous delivery(CI/CD) pipeline that developers use for application code.
- Because the OS and hardware infrastructure is provisioned automatically and the application is encapsulated atop it, these technologies prove complementary for diverse deployment targets, such as test, staging and production.

Benefits of IaC

- Despite its benefits, infrastructure as code poses potential disadvantages. It requires additional tools, such as a configuration management system, that could introduce learning curves and room for error.
- If administrators change server configurations outside of the set infrastructure-as-code template, there is potential for configuration drift. It's important to fully integrate infrastructure as code into systems administration, IT operations and DevOps practices with well-documented policies and procedures.

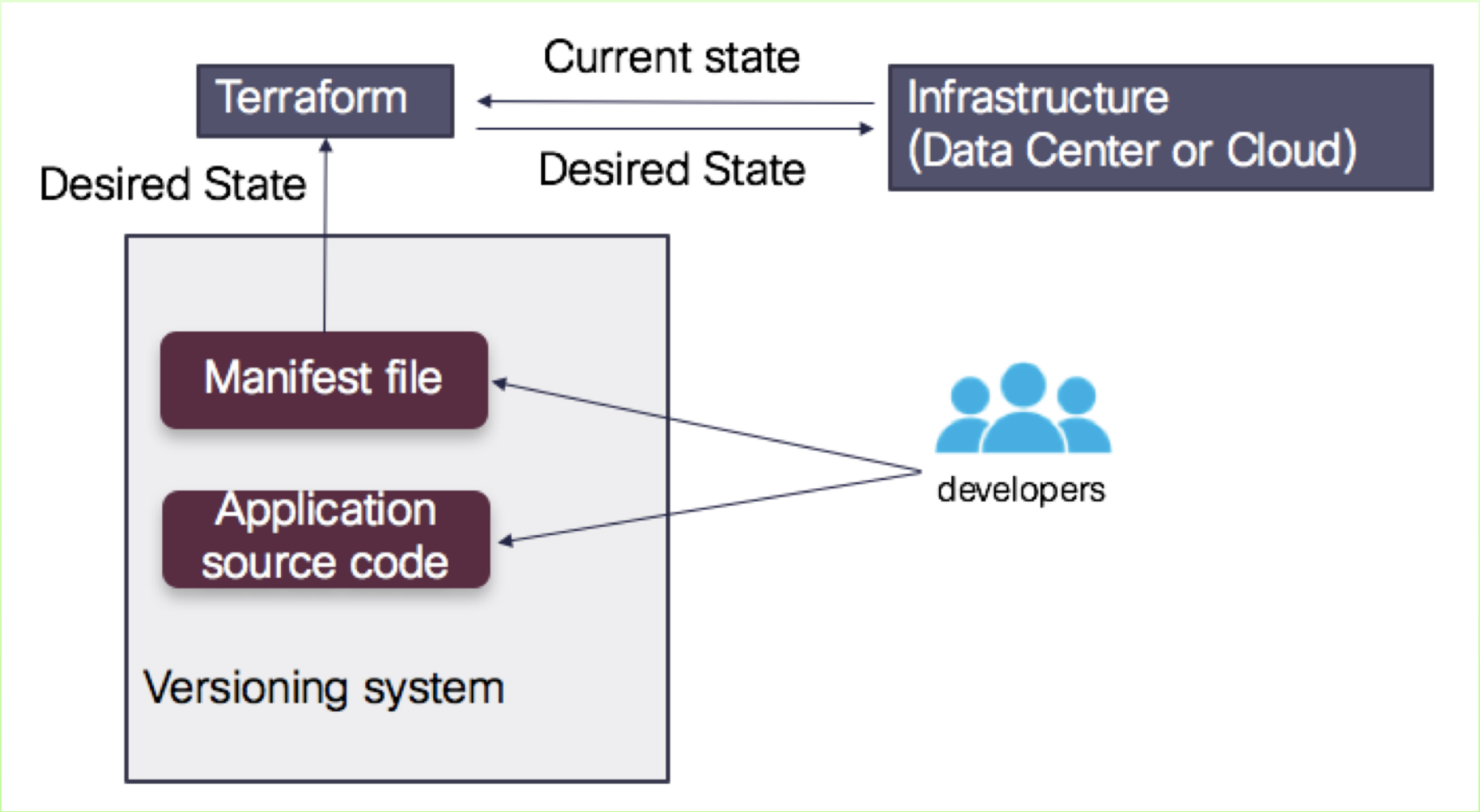
What is Terraform?

- 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?

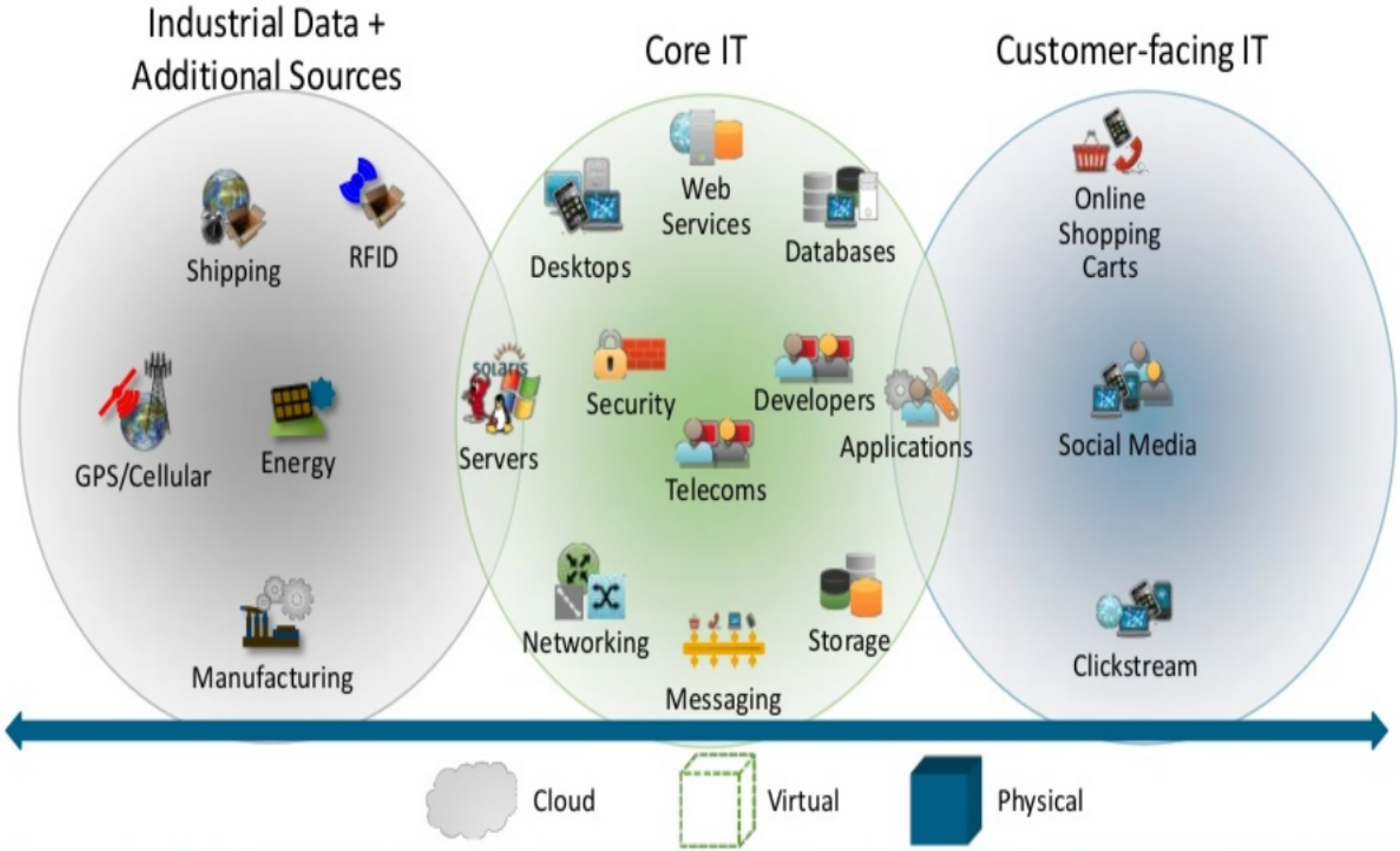


ATGEN

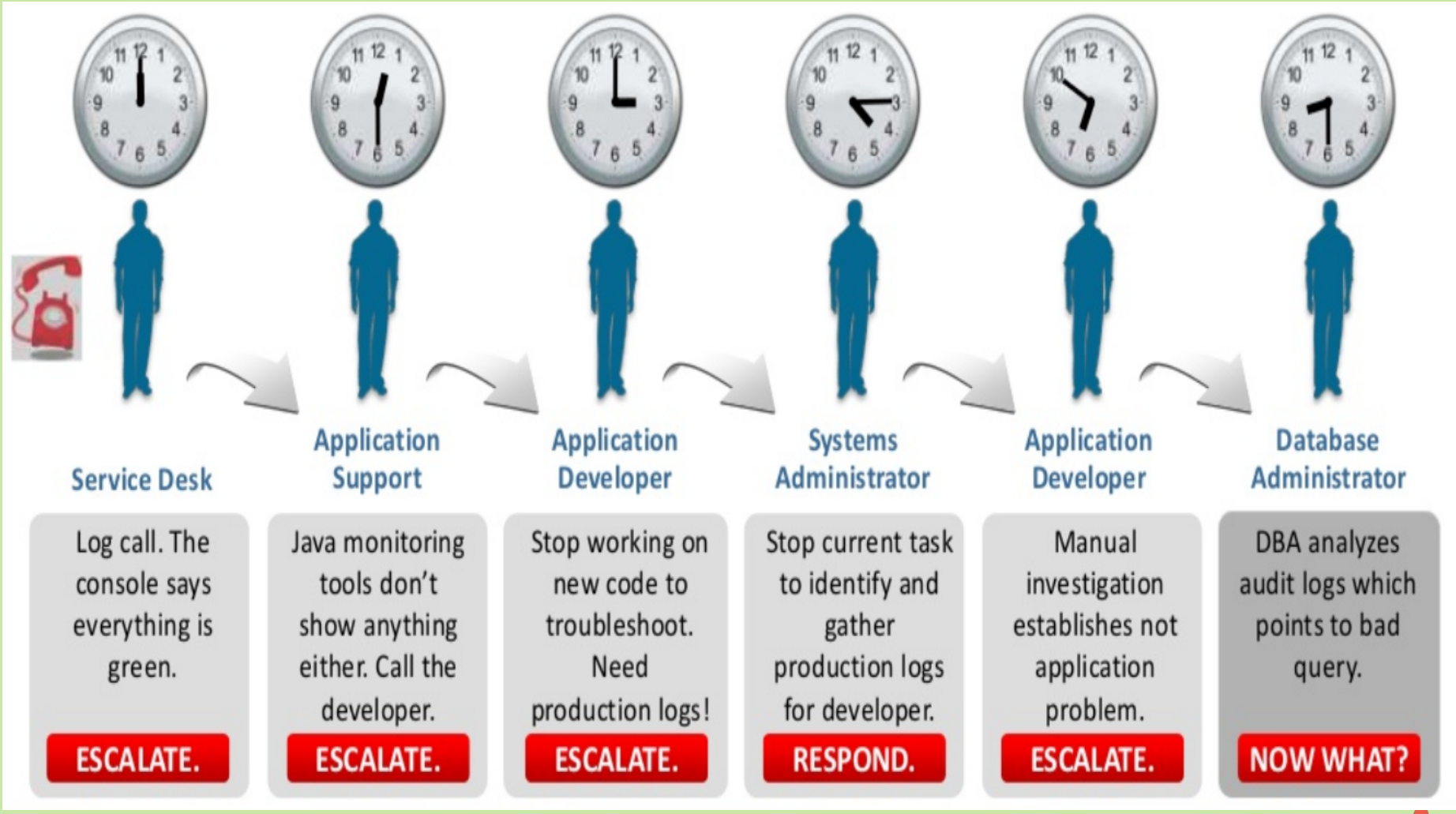
Session: 5

SRE - Splunk

Enterprise Data is Machine Generated

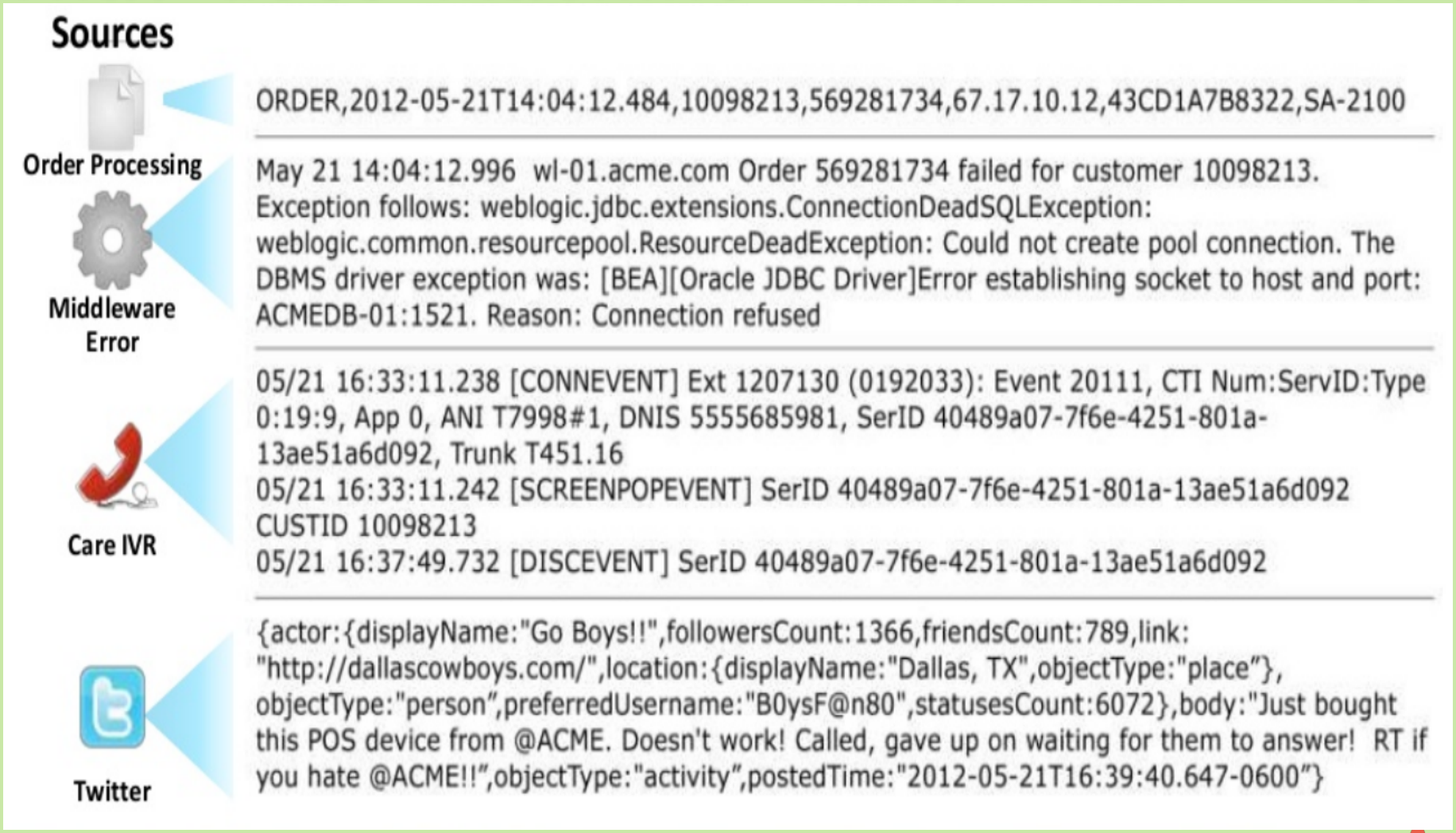


What It's like in the Trenches



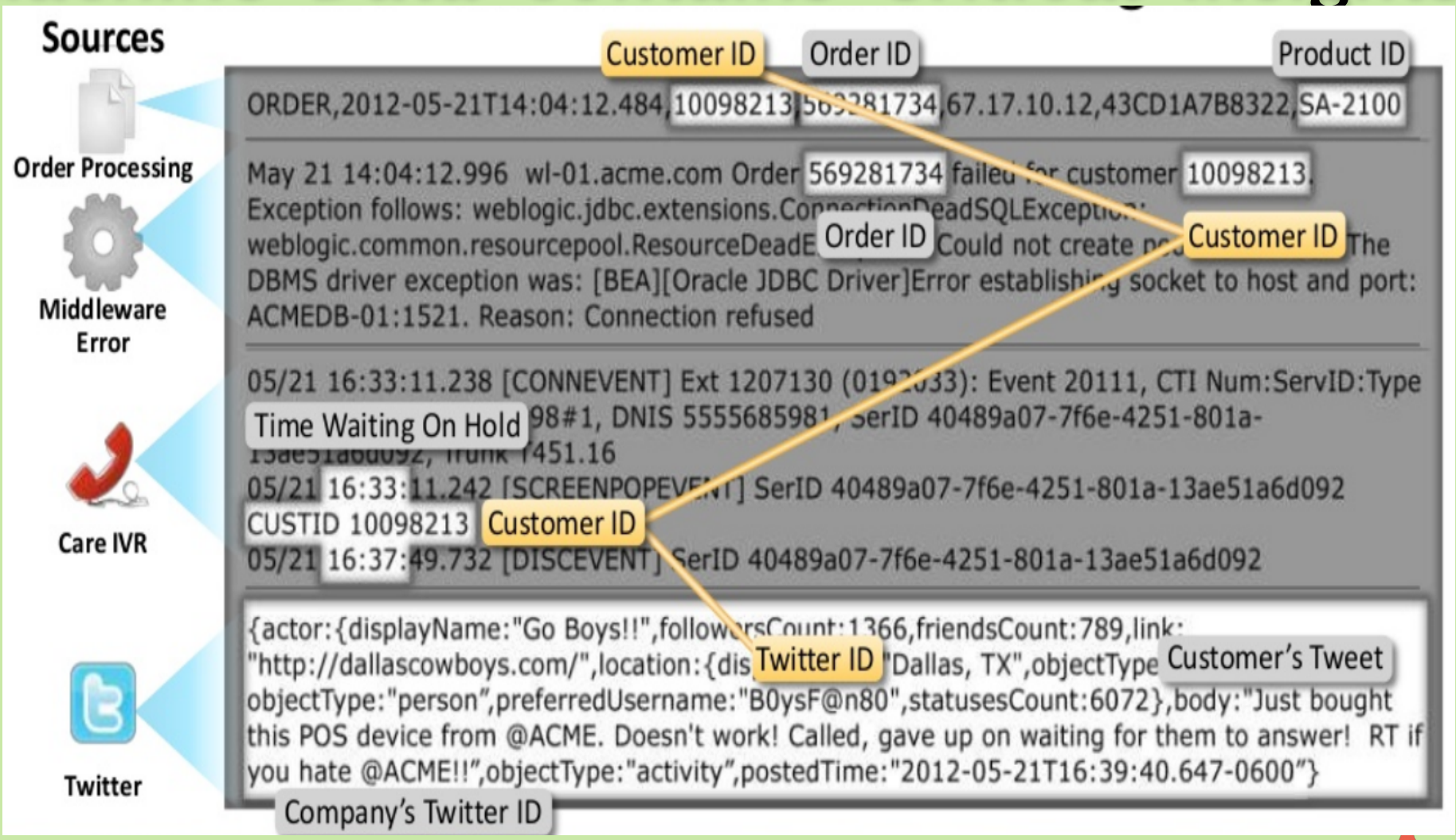
ATGEN

What does Machine Data Look Like?



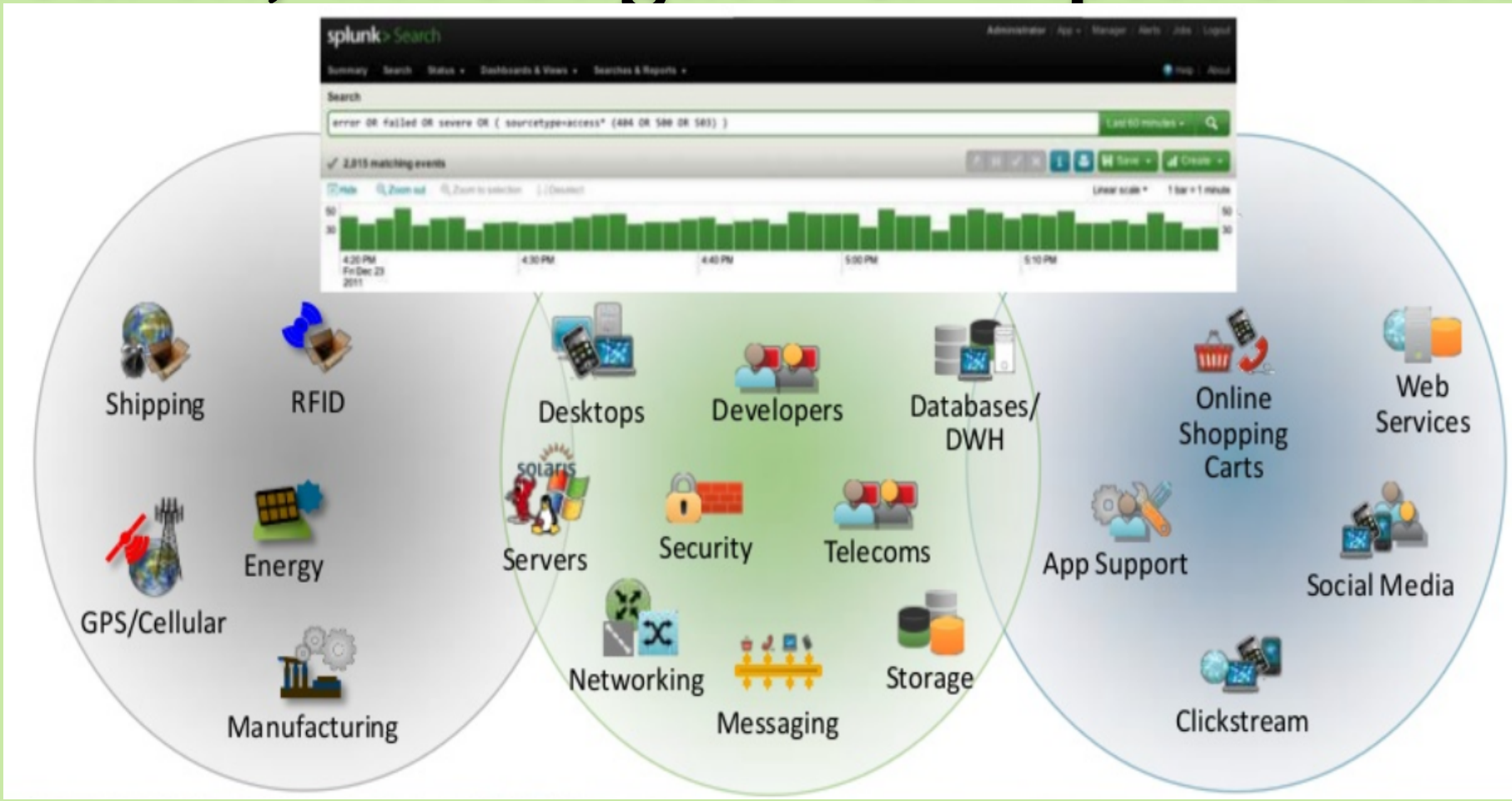
ATGEN

Machine Data contains Critical Insights



ATGEN

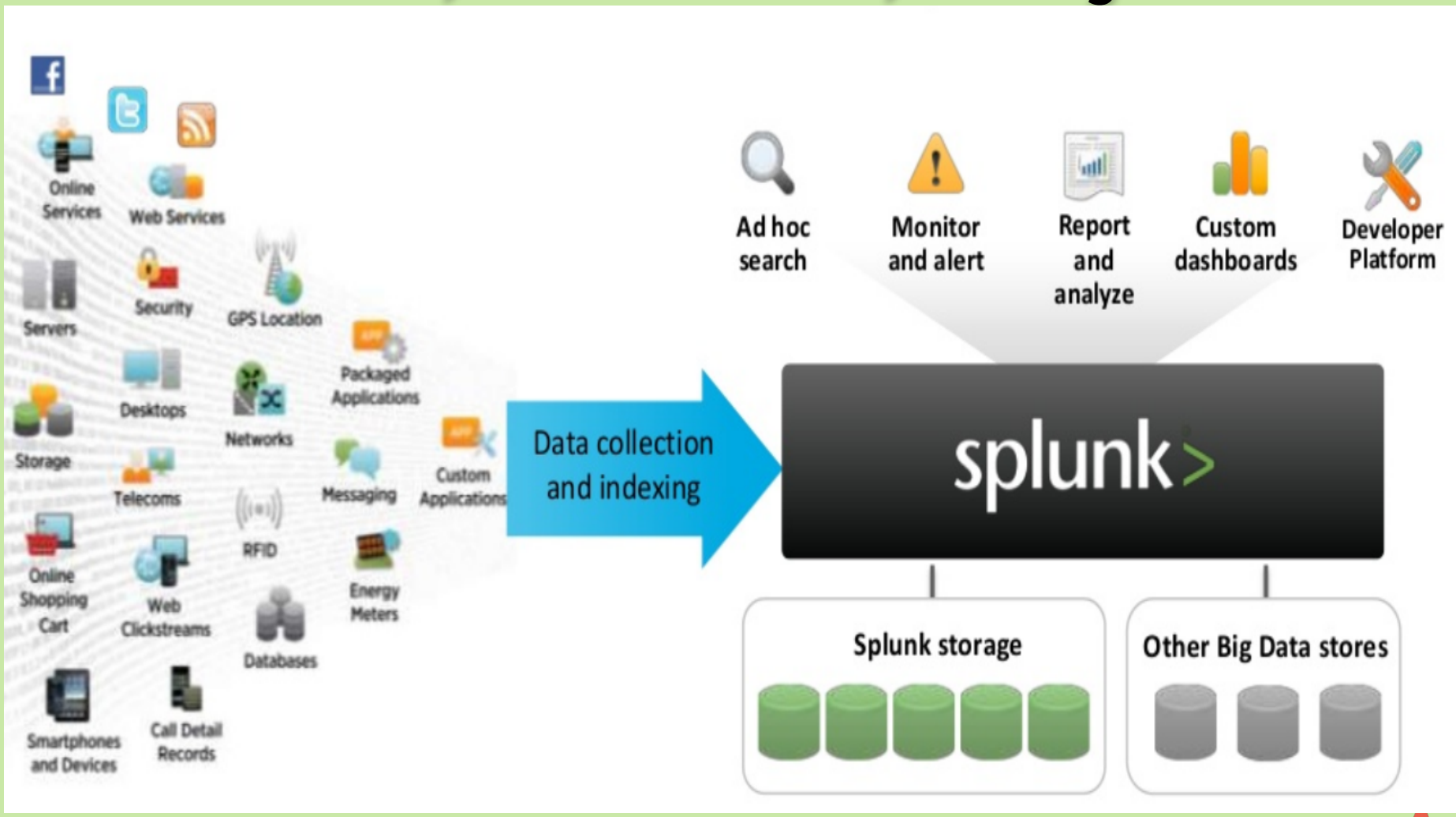
Search, Investigate & Explore Data



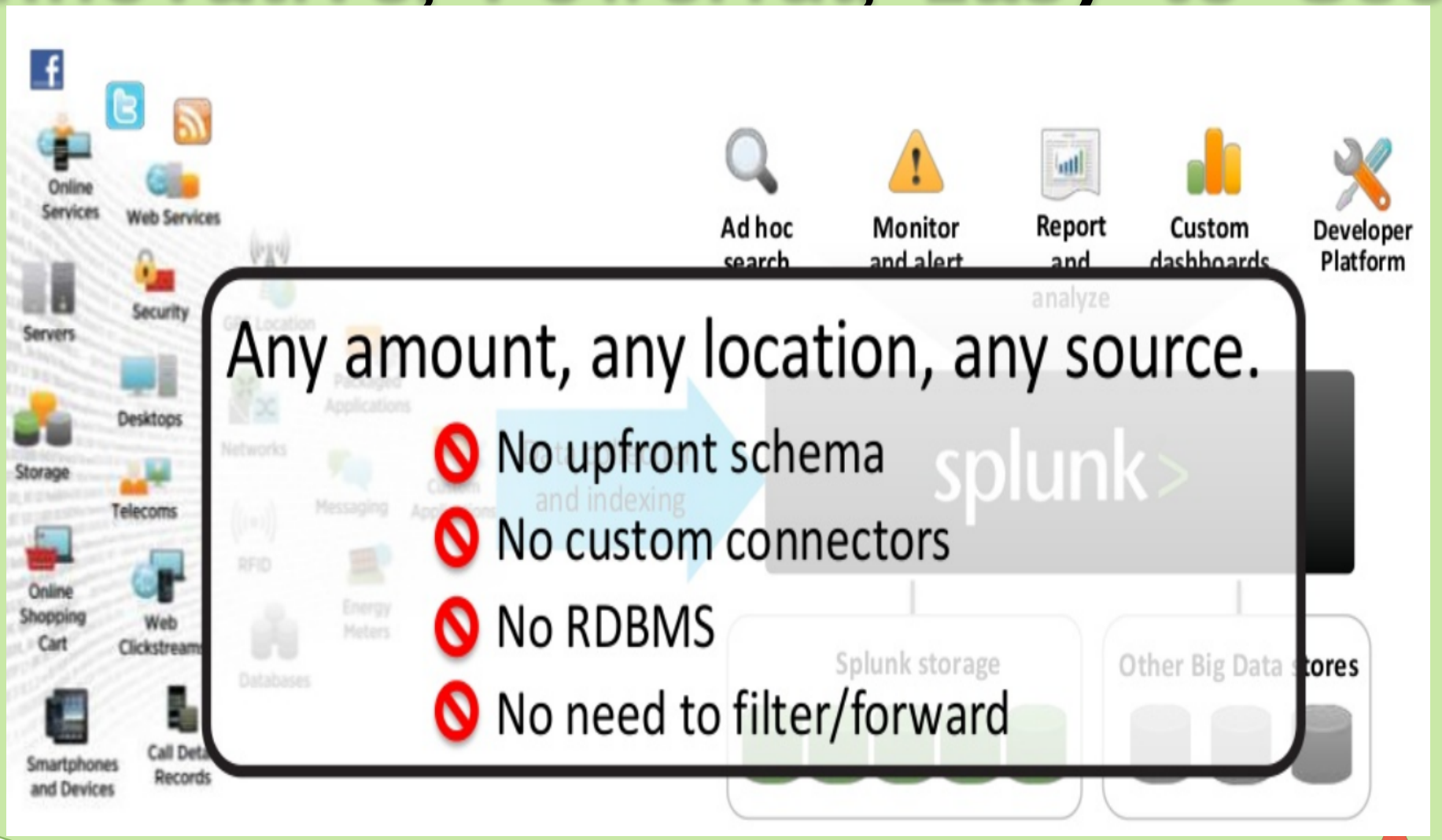
"Find & Fix issues/incidents dramatically faster"

ATGEN

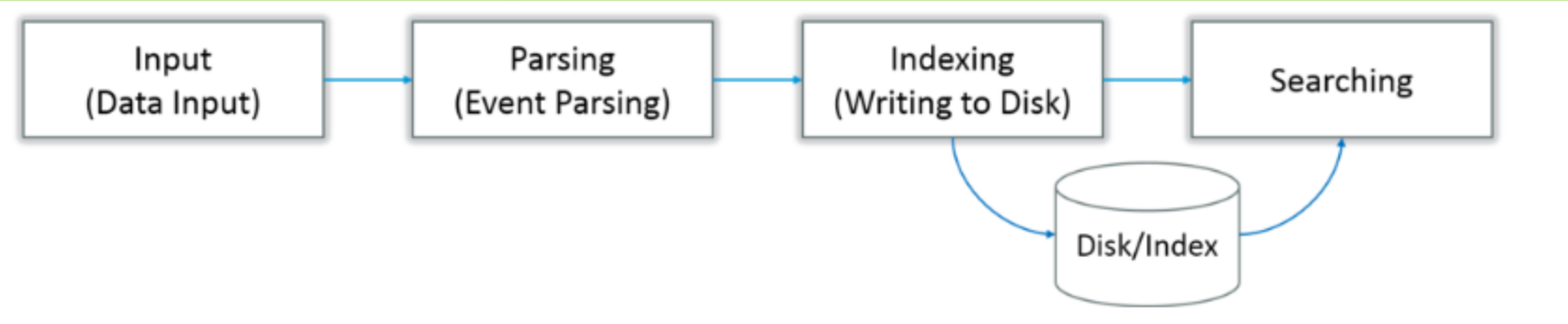
Innovative, Powerful, Easy to Use



Innovative, Powerful, Easy to Use

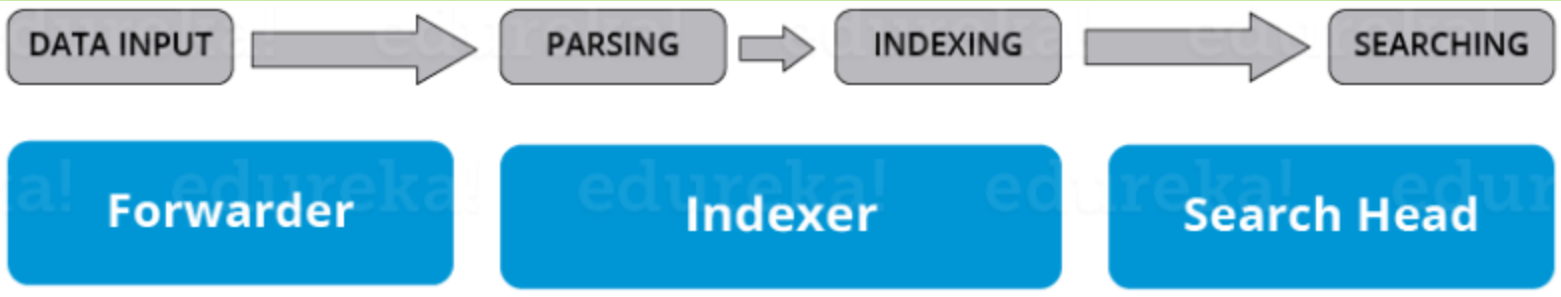


Different Stages in Data Pipeline



- Data Input Stage
  - Splunk software consumes the raw data stream from its source, breaks it into 64K blocks, and annotates each block with metadata keys.
- Data Storage Stage
  - Data storage consists of two phases: Parsing and Indexing.
- Data Searching Stage
  - This stage controls how the user accesses, views, and uses the indexed data.

Splunk Components



- Splunk Forwarder, used for data forwarding
- Splunk Indexer, used for Parsing and Indexing the data
- Search Head, is a GUI used for searching, analyzing and reporting

Splunk Forwarder

- Splunk Forwarder is the component which you have to use for collecting the logs. Suppose, you want to collect logs from a remote machine, then you can accomplish that by using Splunk's remote forwarders which are independent of the main Splunk instance.
- Universal Forwarder
  - It is a simple component which performs minimal processing on the incoming data streams before forwarding them to an indexer.
- Heavy Forwarder
  - Heavy Forwarder typically does parsing and indexing at the source and also intelligently routes the data to the Indexer saving on bandwidth and storage space.

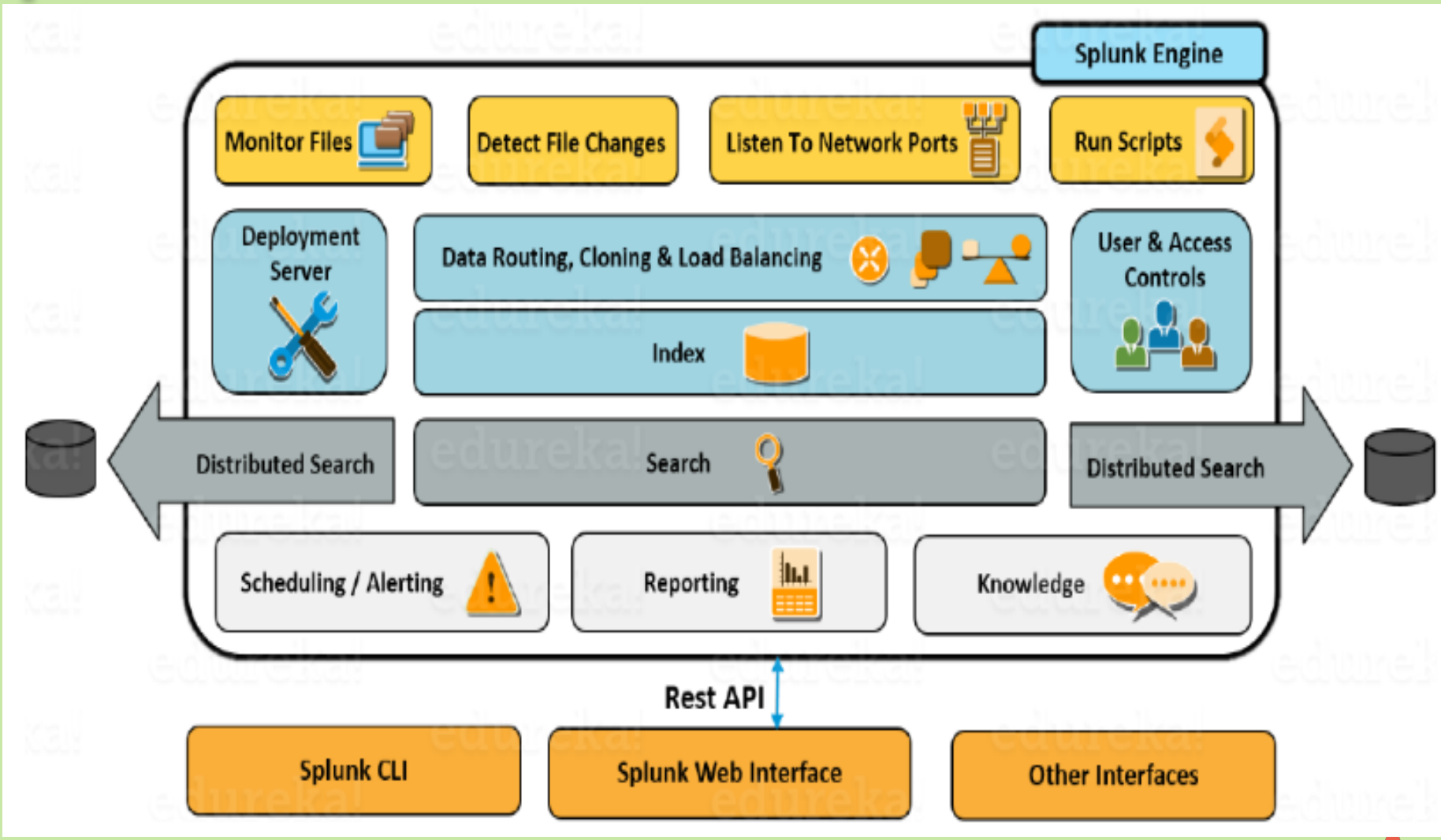
Splunk Indexer

- Splunk instance transforms the incoming data into events and stores it in indexes for performing search operations efficiently.
- If you are receiving the data from a Universal forwarder, then the indexer will first parse the data and then index it.
- If you are receiving the data from a Heavy forwarder, the indexer will only index the data.
- As the Splunk instance indexes your data, it creates a number of files. These files contain one of the below:
  - Raw data in compressed form
  - Indexes that point to raw data (index files, also referred to as tsidx files), plus some metadata files
- These files reside in sets of directories called buckets.
- This indexing process is also known as event processing.

Splunk Search Head

- Search head provides a graphical user interface to users for performing various operations.
- You can search and query the data stored in the Indexer by entering search words and you will get the expected result.
- You can install the search head on separate servers or with other Splunk components on the same server.
- There is no separate installation file for search head, you just have to enable “splunkweb” service on the Splunk server to enable it.
- In a Splunk instance, a search head can send search requests to a group of indexers which perform the actual searches on their indexes. The search head then merges the results and sends them back to the user. This is a faster technique to search data called distributed searching.

Splunk Architecture



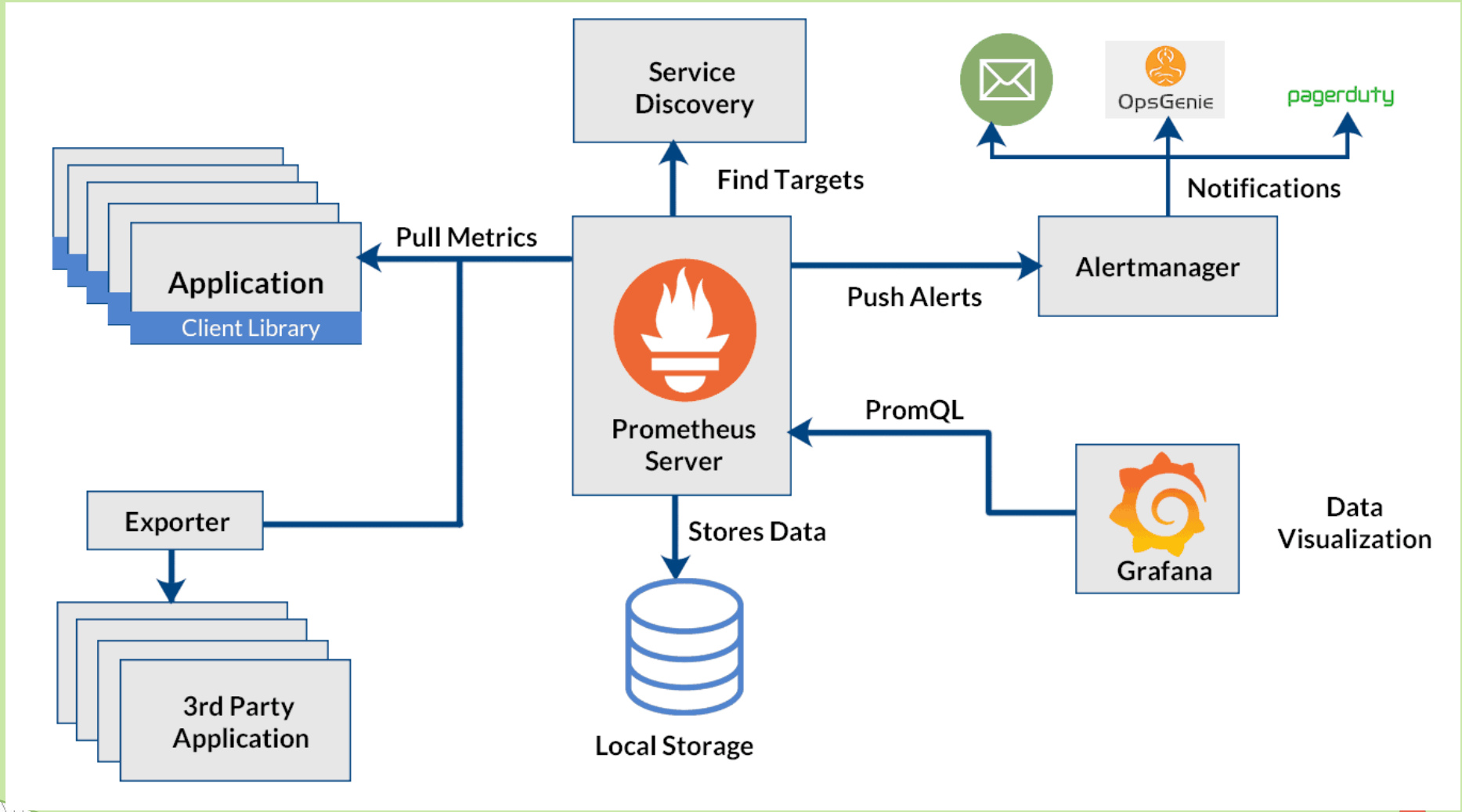
Session: 6

SRE - Prometheus & Grafana

Prometheus/Grafana

- The combination of OSS grafana and Prometheus is becoming a more and more common monitoring stack used by DevOps teams for storing and visualizing time series data.
- Prometheus acts as the storage backend and open source grafana as the interface for analysis and visualization.
- Prometheus collects metrics from monitored targets by scraping metrics from HTTP endpoints on these targets.

Prometheus/Grafana Architecture



ATGEN

Contact us

- Contact @ https://www.atgensoft.com/
- Linkedin: @atgensoft
- FaceBook: @atgensoft
- YouTube: @atgensoft
- Email: SAGAR.MEHTA@ATGENSOFT.COM
- Doc: https://atgensoft.com/learning/download.html
- Learning Code: **jpmc-sre-20062025**
- Quiz: https://forms.gle/Qqh7bL5wwUfUmNct6

Thank You !!