

*Create reliable systems.
Is the game worth the
candle?*

*Alexander Polomodov
Tinkoff*





Hello!

My name is Alexander

- ❖ Technical Director @ Tinkoff
- ❖ Responsible for architecture and delivery management in the company

What are we talking about today



- ❖ *Why is reliability often forgotten?*

What are we talking about today



- ❖ *Why is reliability often forgotten?*
- ❖ *How to choose appropriate risk and control it?*

What are we talking about today



- ❖ *Why is reliability often forgotten?*
- ❖ *How to choose appropriate risk and control it?*
- ❖ *How to design reliable apps?*

What are we talking about today



- ❖ *Why is reliability often forgotten?*
- ❖ *How to choose appropriate risk and control it?*
- ❖ *How to design reliable apps?*
- ❖ *How to implement and operate reliable apps?*

What are we talking about today



- ❖ *Why is reliability often forgotten?*
- ❖ *How to choose appropriate risk and control it?*
- ❖ *How to design reliable apps?*
- ❖ *How to implement and operate reliable apps?*
- ❖ *How to create a culture of reliability?*

Some definitions



Some definitions



Availability

It refers to the percentage of time a system is available to users.

Some definitions



Availability

It refers to the percentage of time a system is available to users.

Reliability

It refers to the likelihood that the system will meet a certain level of performance based on user needs within a certain time frame.



Why is reliability often forgotten?

Why is reliability often forgotten?



Invisibility



Reliability, like security, is largely invisible while everything is going well. As a result, if no incidents occur, then investments in these areas seem like unnecessary costs that will not have an immediate impact

Why is reliability often forgotten?



Invisibility

Assessment



It's not practical to achieve perfect reliability or security, you can use risk-based approaches to estimate the costs of negative events, as well as the up-front and opportunity costs of preventing these events.

Why is reliability often forgotten?



Invisibility

Assessment

Evolution



Systems evolve over time, so as you add functionality or scale the solution, the system will become more complex and if you do not reduce the entropy of the system, then at some point the next small change will lead to a large-scale failure



*How to choose appropriate risk
and control it?*

Types of applications



Types of applications



Business critical



These are critical applications for a business. If these applications are unavailable, then the business is down. Highest availability and lowest latency are desired for these applications. These applications could be user-facing or not, and the classification of a business-critical application depends on each business.

More details in the white paper “Deployment Archetypes”

Types of applications



Business
critical

Line of business
applications

These are applications that support running the business. While these applications do not serve customer-facing traffic, they are typically instrumental for supplying data for the business. They often have requirements to finish work by a particular time.



More details in the white paper “Deployment Archetypes”

Types of applications



Business
critical

Line of business
applications

Internal
applications

~~Bad~~
~~Good~~

Best
Effort

These are applications that are for internal consumption for a business. Best effort availability is required. Employees want these to be always available, but if they are not, then the impact on the business is lower.

More details in the white paper “Deployment Archetypes”

Level of service



More details in the book “Site Reliability Engineering”

Level of service



SLI
(service level
indicators)

A carefully defined quantitative measure of some aspect of the level of service that is provided (i.e., request latency, error rate, system throughput, availability, ...)



More details in the book "Site Reliability Engineering"

Level of service



SLI
(service level
indicators)

SLO
(service level
objectives)

A target value or range of values for a service level that is measured by an SLI. A natural structure for SLOs is thus $SLI \leq \text{target}$, or $\text{lower bound} \leq SLI \leq \text{upper bound}$. Choosing an appropriate SLO is complex. Publishing SLOs to users sets expectations about how a service will perform.



More details in the book “Site Reliability Engineering”

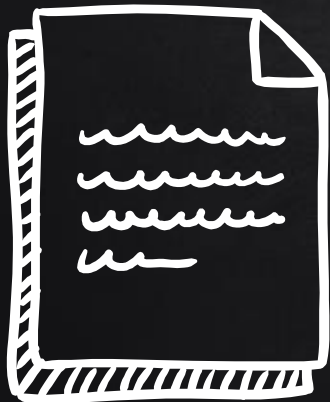
Level of service



SLI
(service level
indicators)

SLO
(service level
objectives)

SLA
(service level
agreements)



An explicit or implicit contract with your users that includes consequences of meeting (or missing) the SLOs they contain. An easy way to tell the difference between an SLO and an SLA is to ask "what happens if the SLOs aren't met?": if there is no explicit consequence, then you are almost certainly looking at an SLO

More details in the book "Site Reliability Engineering"

Monitoring



More details in the book “Site Reliability Engineering”

Monitoring



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

More details in the book “Site Reliability Engineering”

Monitoring



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



White-box
monitoring

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

More details in the book “Site Reliability Engineering”

Monitoring



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



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

Testing externally visible behavior as a user would see it.

More details in the book "Site Reliability Engineering"

Monitoring



The four golden
signals



More details in the book "Site Reliability Engineering"

Monitoring



The four golden signals

- ❖ Latency - the time it takes to service a request

More details in the book "Site Reliability Engineering"

Monitoring



The four golden signals

- ❖ Latency - the time it takes to service a request
- ❖ Traffic - a measure of how much demand is being placed on your system

More details in the book "Site Reliability Engineering"

Monitoring



The four golden signals

- ❖ Latency – the time it takes to service a request
- ❖ Traffic – a measure of how much demand is being placed on your system
- ❖ Errors – the rate of requests that fail, either explicitly, implicitly, or by policy

More details in the book “Site Reliability Engineering”

Monitoring



The four golden signals

- ❖ Latency – the time it takes to service a request
- ❖ Traffic – a measure of how much demand is being placed on your system
- ❖ Errors – the rate of requests that fail, either explicitly, implicitly, or by policy
- ❖ Saturation – how "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained

More details in the book "Site Reliability Engineering"

Monitoring



The four golden signals

RED method

- ❖ Latency – the time it takes to service a request.
- ❖ Traffic – a measure of how much demand is being placed on your system
- ❖ Errors – the rate of requests that fail, either explicitly, implicitly, or by policy
- ❖ Saturation – how "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained

Monitoring



The four golden signals

- ❖ Latency – the time it takes to service a request.
- ❖ Traffic – a measure of how much demand is being placed on your system
- ❖ Errors – the rate of requests that fail, either explicitly, implicitly, or by policy
- ❖ Saturation – how "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained

RED method

- ❖ Rate – the number of requests, per second

Monitoring



The four golden signals

- ❖ Latency - the time it takes to service a request.
- ❖ Traffic - a measure of how much demand is being placed on your system
- ❖ Errors - the rate of requests that fail, either explicitly, implicitly, or by policy
- ❖ Saturation - how "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained

RED method

- ❖ Rate - the number of requests, per second
- ❖ Errors - the number of failed requests per second

Monitoring



The four golden signals

- ❖ Latency - the time it takes to service a request.
- ❖ Traffic - a measure of how much demand is being placed on your system
- ❖ Errors - the rate of requests that fail, either explicitly, implicitly, or by policy
- ❖ Saturation - how "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained

RED method

- ❖ Rate - the number of requests, per second
- ❖ Errors - the number of failed requests per second
- ❖ Duration - distributions of the amount of time each request takes



How to design reliable apps?

More details in the white paper “Deployment Archetypes for Cloud Applications”

Important concepts for reliability



More details in the white paper “Deployment Archetypes for Cloud Applications”

Important concepts for reliability



Fault domains

A key part of the design is how the application reasons about fault domains and how it provides redundancy and scales across those fault domains to maximize availability. A fault domain is a set of infrastructure parts that together represent a single point of failure.

More details in the white paper “Deployment Archetypes for Cloud Applications”

Important concepts for reliability



Fault
domains

Sharding

Applications may apply sharding across their users or data so that they are served across different fault domains. In this way, an issue with one fault domain will only impact a subset of the users/data, thus containing the failure radius (often called the blast radius).

More details in the white paper “Deployment Archetypes for Cloud Applications”

Important concepts for reliability



Fault
domains

Sharding

Incremental
rollout

Similarly, code and configuration changes should be rolled out incrementally across the different fault domains to gradually introduce a change into production, with the ability to quickly roll back if any production issues are discovered to return the application to a healthy state. This allows code and configuration production issues to be discovered early on and reduces the impact.

More details in the white paper “Deployment Archetypes for Cloud Applications”

Important concepts for reliability



Fault
domains

Sharding

Incremental
rollout

Dependency
management

Applications need to understand their dependencies, the availability and failure modes of those dependencies, and to evaluate the multiplicative implications across these dependencies on the application's design and availability. As part of the overall manageability, separating out parts of the application into its vital and non-vital services, identifying the availability targets of each, and continuously improving the vital parts, are important.

More details in the white paper "Deployment Archetypes for Cloud Applications"

Core data concepts related to reliability

More details in the white paper “Deployment Archetypes for Cloud Applications”

Core data concepts related to reliability

Data durability

Long-term data protection, where the stored data does not suffer from corruption and is not lost or compromised. To achieve this, the underlying data storage system often replicates the data and performs error-correcting checks and scrubbing of the data to prevent data decay.

More details in the white paper “Deployment Archetypes for Cloud Applications”

Core data concepts related to reliability

Data durability

Data
availability

Access to data upon request. High data availability is achieved by placing and replicating the data across more than one failure domain, keeping the data durable, The type of replication, asynchronous (eventually consistent) or synchronous (strongly consistent), along with data failover capabilities are important building blocks for achieving data availability.

More details in the white paper “Deployment Archetypes for Cloud Applications”

Core data concepts related to reliability

Data durability

Data
availability

Data backup

Point-in-time snapshots of data. Backups are important for protecting against application and human errors, and they can also be used as a means for disaster recovery. All applications should use backup services to protect against accidental loss or corruption of data due to application-level issues.

More details in the white paper “Deployment Archetypes for Cloud Applications”

Disaster recovery objectives



More details in the white paper “Deployment Archetypes for Cloud Applications”

Disaster recovery objectives



RPO
(Recovery point
objective)

RPO captures how old a copy (backup or replica) of the data is compared to the current state in production.

More details in the white paper “Deployment Archetypes for Cloud Applications”

Disaster recovery objectives



RPO
(Recovery point
objective)

RPO captures how old a copy (backup or replica) of the data is compared to the current state in production.

RTO
(Recovery time
objective)

RTO captures how long it takes to restore an application during recovery to bring it back online and available in production, which includes access to the data needed for running the application.

More details in the white paper “Deployment Archetypes for Cloud Applications”

Disaster recovery objectives



RPO
(Recovery point objective)

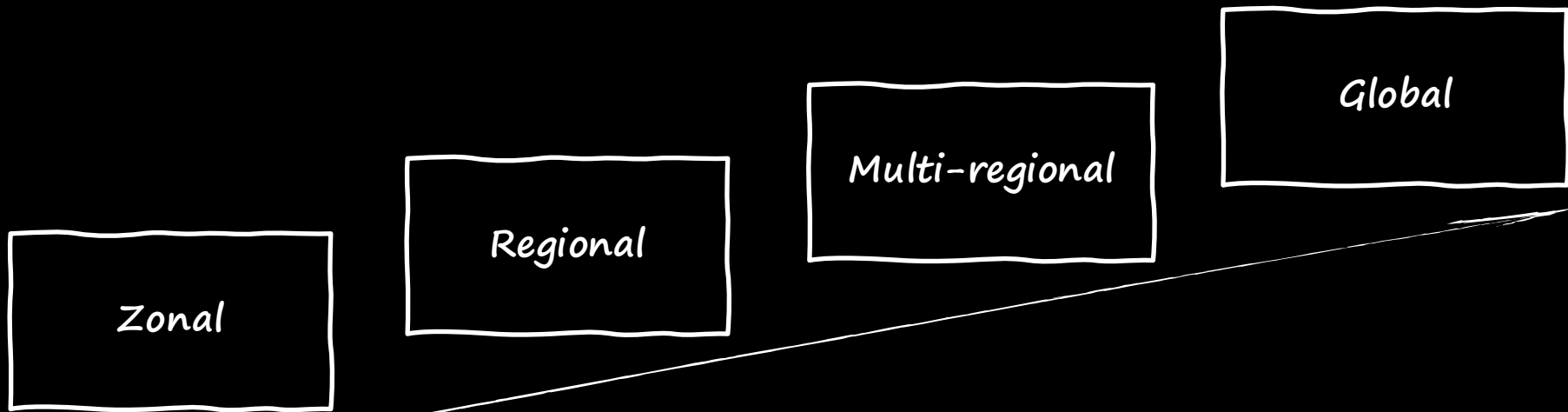
RPO captures how old a copy (backup or replica) of the data is compared to the current state in production.

RTO
(Recovery time objective)

RTO captures how long it takes to restore an application during recovery to bring it back online and available in production, which includes access to the data needed for running the application.

We should choose deployment model according to this objective

More details in the white paper “Deployment Archetypes for Cloud Applications”



Application deployment archetypes

More details in the white paper “Deployment Archetypes for Cloud Applications”

Zonal



```
graph TD; Zonal[Zonal] --- Box[All components of an application run within a single zone. A zone provides a set of clusters with the infrastructure needed to run services (compute, storage, networking, data, etc.) within that zone.];
```

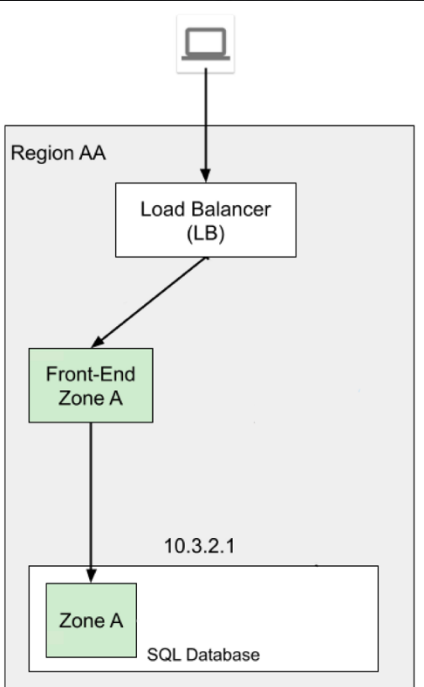


All components of an application run within a single zone. A zone provides a set of clusters with the infrastructure needed to run services (compute, storage, networking, data, etc.) within that zone.

Zonal



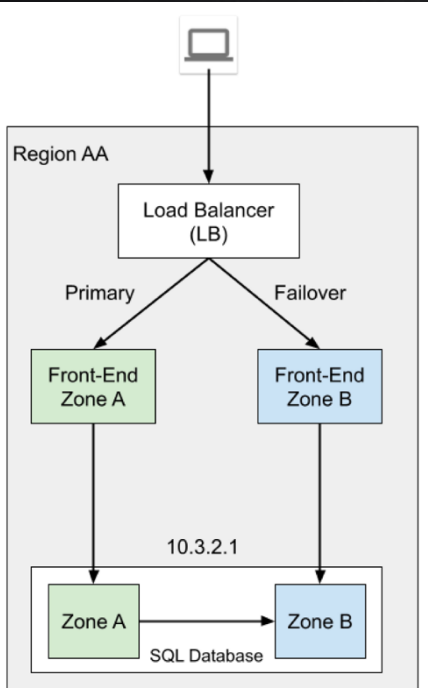
❖ Single Zone



Running an application only within a single zone is not targeted toward high availability, since a zone is considered as a single failure domain from both software issues as well as other types of disasters (e.g., fire). Even so, applications that need supercomputer-like connectivity, as well as applications that do not need high availability, leverage single-zone deployments.

Zonal

- ❖ *Single Zone*
- ❖ *Single Zone with failover*



If the primary zone has issues, then the recovery zone is used to start the application up again. Many enterprise applications are built to run in a form of primary/failover configuration, also known as Highly Available (HA) topologies, and this is an established pattern used in enterprise and on-premises deployments over the years.

Zonal

Regional

- ❖ Single Zone
- ❖ Single Zone with failover

All components of an application are deployed and run out of one Cloud region. A region consists of 3 or more zones, where each zone is treated as a separate fault domain. High availability can be achieved by replicating the application across zones within the region. These applications are typically designed to run with a data store that shares data and makes it accessible across that region.

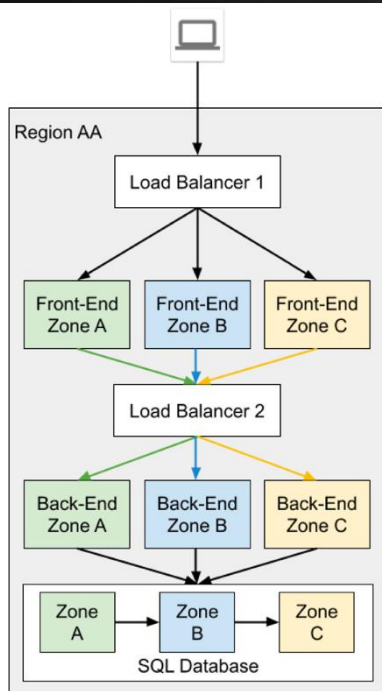


Zonal

Regional

- ❖ Single Zone
- ❖ Single Zone with failover

- ❖ Single Region



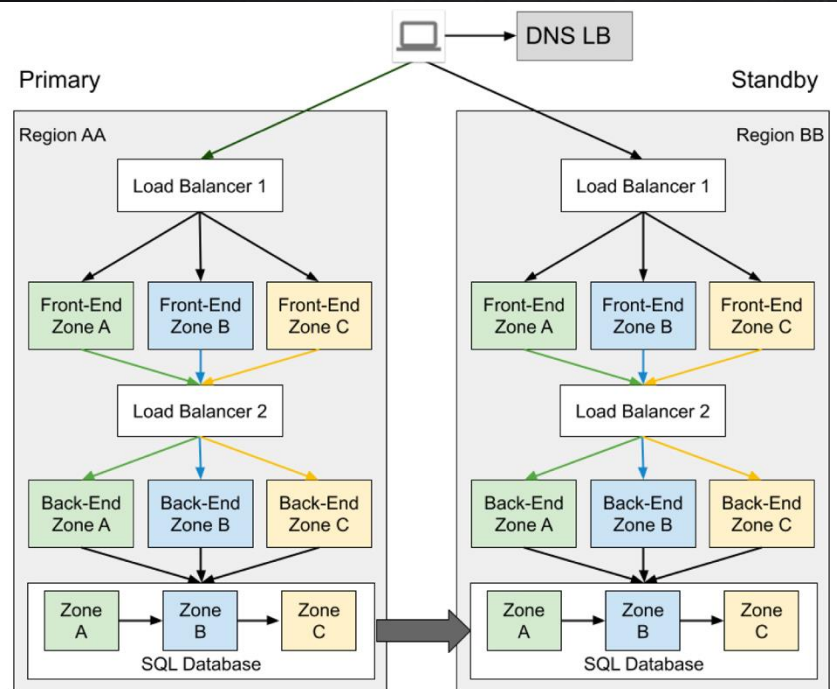
Running an application in a region means running an application spread across multiple zones within that region, where each zone is treated as an independent failure domain. A best practice here is to replicate the application across all of the zones within the region and keep the size of each deployment approximately the same across zones. This ensures the application always has capacity available in other zones when there is a zonal failure.

Zonal

Regional

- ❖ Single Zone
- ❖ Single Zone with failover

- ❖ Single Region
- ❖ Single Region with failover



Application data is synchronously replicated within a primary region, providing $RPO = 0$ for in-region failures. It is also asynchronously replicated to a standby region that is sufficiently distant from the primary region



Zonal

Regional

Multi-regional

- ❖ Single Zone
- ❖ Single Zone with failover

- ❖ Single Region
- ❖ Single Region with failover

The application serving stack runs and is stitched together across multiple regions to achieve higher availability and low end-user latency through geographic distribution. In this deployment archetype, data is typically replicated and shared across regions. This archetype is commonly used for applications that want to achieve high availability, such as user-facing applications.

Zonal

Regional

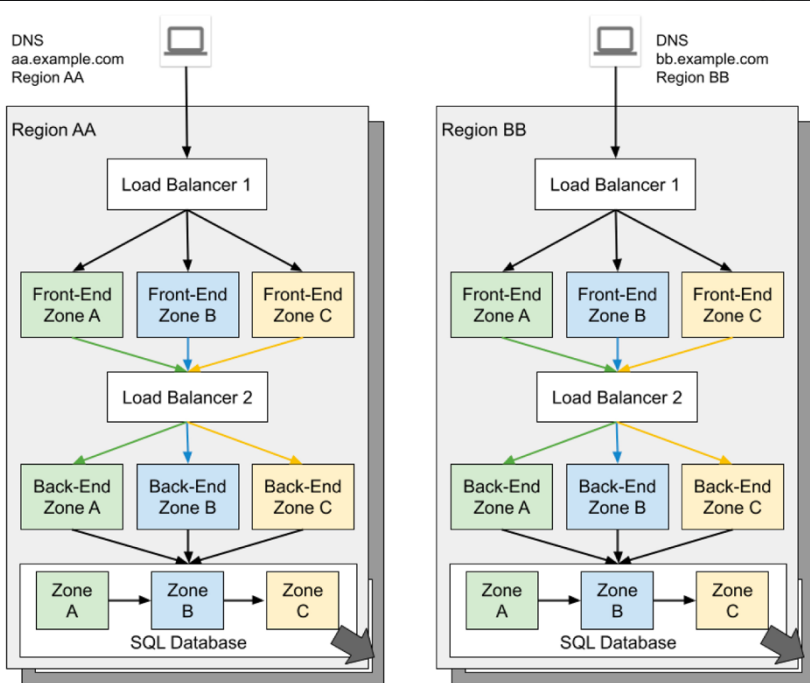
Multi-regional



- ❖ Single Zone
- ❖ Single Zone with failover

- ❖ Single Region
- ❖ Single Region with failover

- ❖ Isolated regional stacks with data sharding



If the application data is partitionable into separate databases, then one deployment option some applications have considered is to partition or shard the application and data across multiple regions into separate isolated stacks. In this approach, each stack would use the Primary Region with Failover Region approach, and a given user's data is confined to a single regional deployment based on the sharding.



Zonal

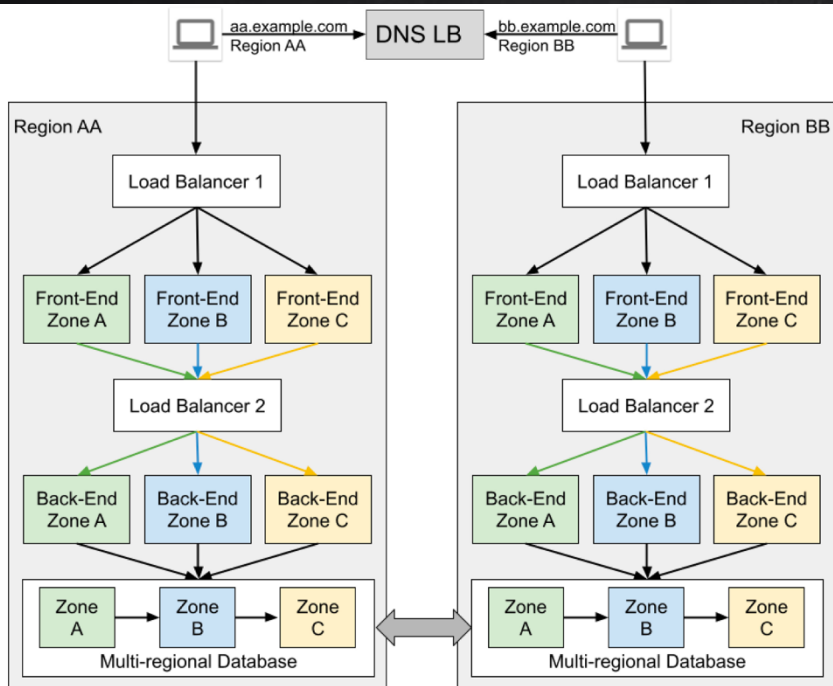
Regional

Multi-regional

- ❖ Single Zone
- ❖ Single Zone with failover

- ❖ Single Region
- ❖ Single Region with failover

- ❖ Isolated regional stacks with data sharding
- ❖ DNS LB with Isolated regional stacks



Add to Isolated stack with sharding DNS Load Balancer to connect regional application stacks and load balance traffic across the regions. This is for applications that can run across multiple regions and be run with a data store that shares data and makes it accessible across those regions. For this deployment model, multi-regional or global storage and database solutions should be used, since the same data needs to be accessible at the same time across multiple regions.

Zonal

- ❖ Single Zone
- ❖ Single Zone with failover

Regional

- ❖ Single Region
- ❖ Single Region with failover

Multi-regional

- ❖ Isolated regional stacks with data sharding
- ❖ DNS LB with isolated regional stacks

Global

The application stack is spread and replicated across Cloud regions around the globe and data is available worldwide via global databases and storage. Applications consisting of a large number of services and microservices benefit from this deployment archetype.

Zonal

- ❖ Single Zone
- ❖ Single Zone with failover

Regional

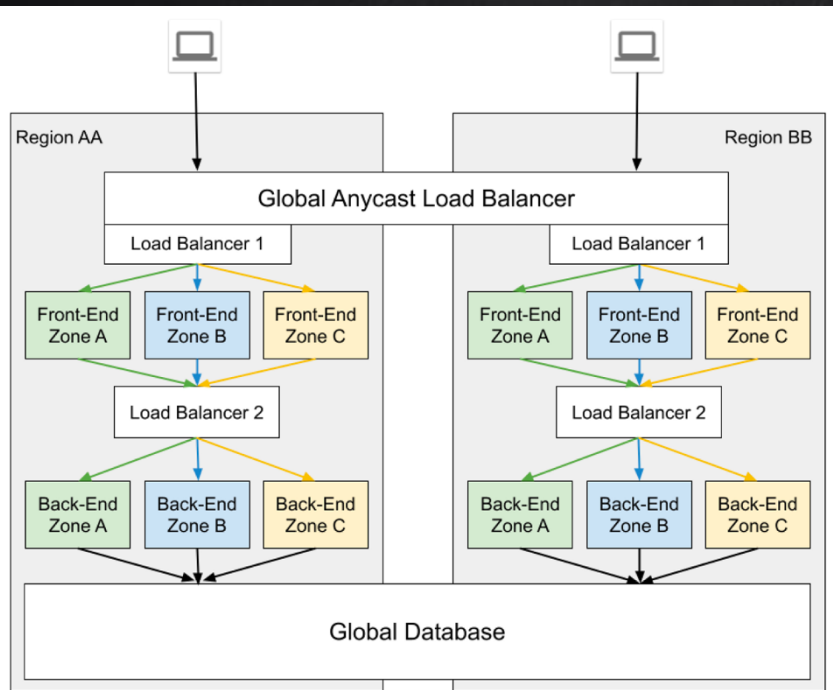
- ❖ Single Region
- ❖ Single Region with failover

Multi-regional

- ❖ Isolated regional stacks with data sharding
- ❖ DNS LB with isolated regional stacks

Global

- ❖ Global anycast LB with isolated stacks



In this deployment model, a Global anycast LB ingests traffic and then sends traffic to the regional LB in the region containing the application owner's compute resources, depending on geo-mapping, health and weights.

Zonal

Regional

Multi-regional

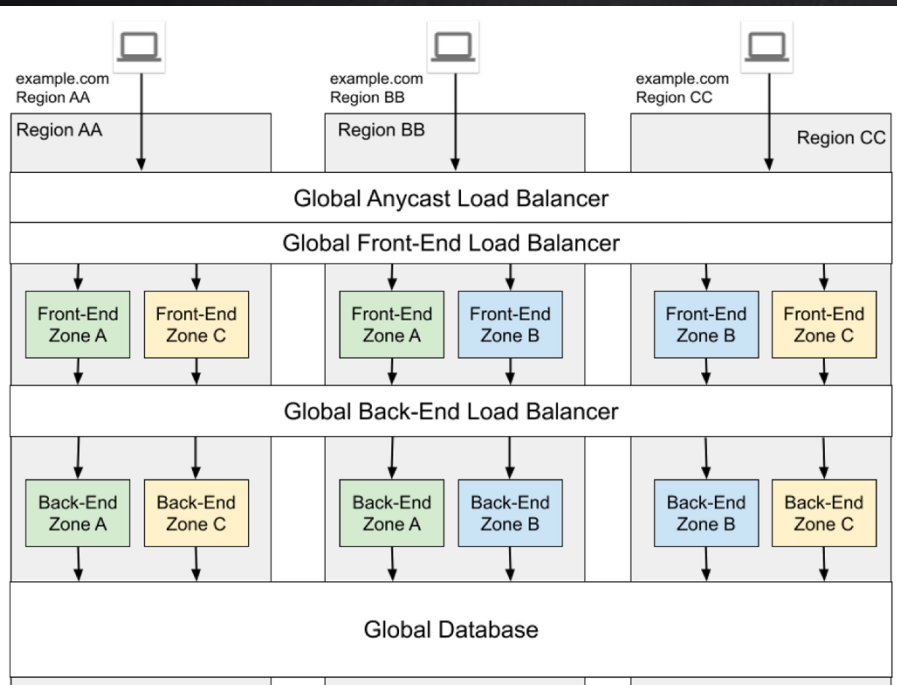
Global

- ❖ Single Zone
- ❖ Single Zone with failover

- ❖ Single Region
- ❖ Single Region with failover

- ❖ Isolated regional stacks with data sharding
- ❖ DNS LB with isolated regional stacks

- ❖ Global anycast LB with isolated stacks
- ❖ Global services stack



In this deployment, services are global. The data is also global and synchronously or asynchronously replicated and available in all regions where services run (e.g., using Google Spanner). In addition, having a global network is important to making a Global Services Stack possible.

Zonal

- ❖ Single Zone
- ❖ Single Zone with failover

Regional

- ❖ Single Region
- ❖ Single Region with failover

Multi-regional

- ❖ Isolated regional stacks with data sharding
- ❖ DNS LB with isolated regional stacks

Global

- ❖ Global anycast LB with isolated stacks
- ❖ Global services stack

Increasing availability

Zonal

- ❖ Single Zone
- ❖ Single Zone with failover

Regional

- ❖ Single Region
- ❖ Single Region with failover

Multi-regional

- ❖ Isolated regional stacks with data sharding
- ❖ DNS LB with Isolated regional stacks

Global

- ❖ Global anycast LB with isolated stacks
- ❖ Global services stack

Increasing availability
and cost

Design principles



❖ Design Tradeoffs

More details in the book “Building Secure and Reliable Systems”

Design principles



- ❖ *Design Tradeoffs*
- ❖ *Design for Understandability*

More details in the book “Building Secure and Reliable Systems”

Design principles



- ❖ *Design Tradeoffs*
- ❖ *Design for Understandability*
- ❖ *Design for a Changing Landscape*

More details in the book “Building Secure and Reliable Systems”

Design principles



- ❖ *Design Tradeoffs*
- ❖ *Design for Understandability*
- ❖ *Design for a Changing Landscape*
- ❖ *Design for Resilience*

More details in the book “Building Secure and Reliable Systems”

Design principles



- ❖ *Design Tradeoffs*
- ❖ *Design for Understandability*
- ❖ *Design for a Changing Landscape*
- ❖ *Design for Resilience*
- ❖ *Design for Recovery*

More details in the book “Building Secure and Reliable Systems”

Design principles



- ❖ *Design Tradeoffs*
- ❖ *Design for Understandability*
- ❖ *Design for a Changing Landscape*
- ❖ *Design for Resilience*
- ❖ *Design for Recovery*
- ❖ *Designing for Defense (Mitigating Denial-of-Service Attacks)*

More details in the book “Building Secure and Reliable Systems”



*How to implement and
operate reliable apps?
(engineering practices)*

More details in the book "Accelerate"

Focus on a global outcome to ensure teams are cooperative

Key characteristics of measuring performance

Focus on outcomes not output

Measures that meet these criteria

Software delivery performance tempo
(from lean management: lead time and batch size)

Delivery lead time

There are 2 parts of **lead time** (discovery and delivery). Product delivery lead time - time it takes to go from code committed to code successfully running in production

Deployment frequency

Deployment frequency works as a proxy for **batch size** since it's easy to measure and typically has low variability

Software stability

MTTR
(mean time to recover)

How quickly can service be restored?

Change fail rate

A key metric when making changes to systems is what percentage of changes to production fail.

There is no trade-off between improving performance and stability and quality



Key principles of continuous delivery

Build quality in

In continuous delivery, we invest in building a culture supported by tools and people where we can detect any issues quickly, so that they can be fixed straight away when they are cheap to detect and resolve

Work in small batches

Even though working in small chunks adds some overhead, it reaps enormous rewards by allowing us to avoid work that delivers zero or negative value for organization. A key goal of CD is changing the economics of the software delivery process so the cost of pushing out individual changes is very low

Computers perform repetitive tasks; people solve problems

Invest in simplifying and automating repetitive work that takes a long time, such as regression testing and software deployments

Relentlessly pursue continuous improvement

High performing teams are never satisfied, they make improvement part of everybody's daily work

Everyone is responsible

A key objective for management is making the state of system-level outcome (throughput, quality, stability) transparent, working with the rest of organization to set measurable, achievable, time-bound goals for these outcomes, and then helping their teams work toward them

Engineering practices

Test automation

Continuous integration

Deployment automation

Version control

Trunk-based development

Test data management

Shift left on security

Monitoring

Loosely coupled architecture

Proactive notifications

Empowered teams

Continuous delivery

Westrum organizational culture

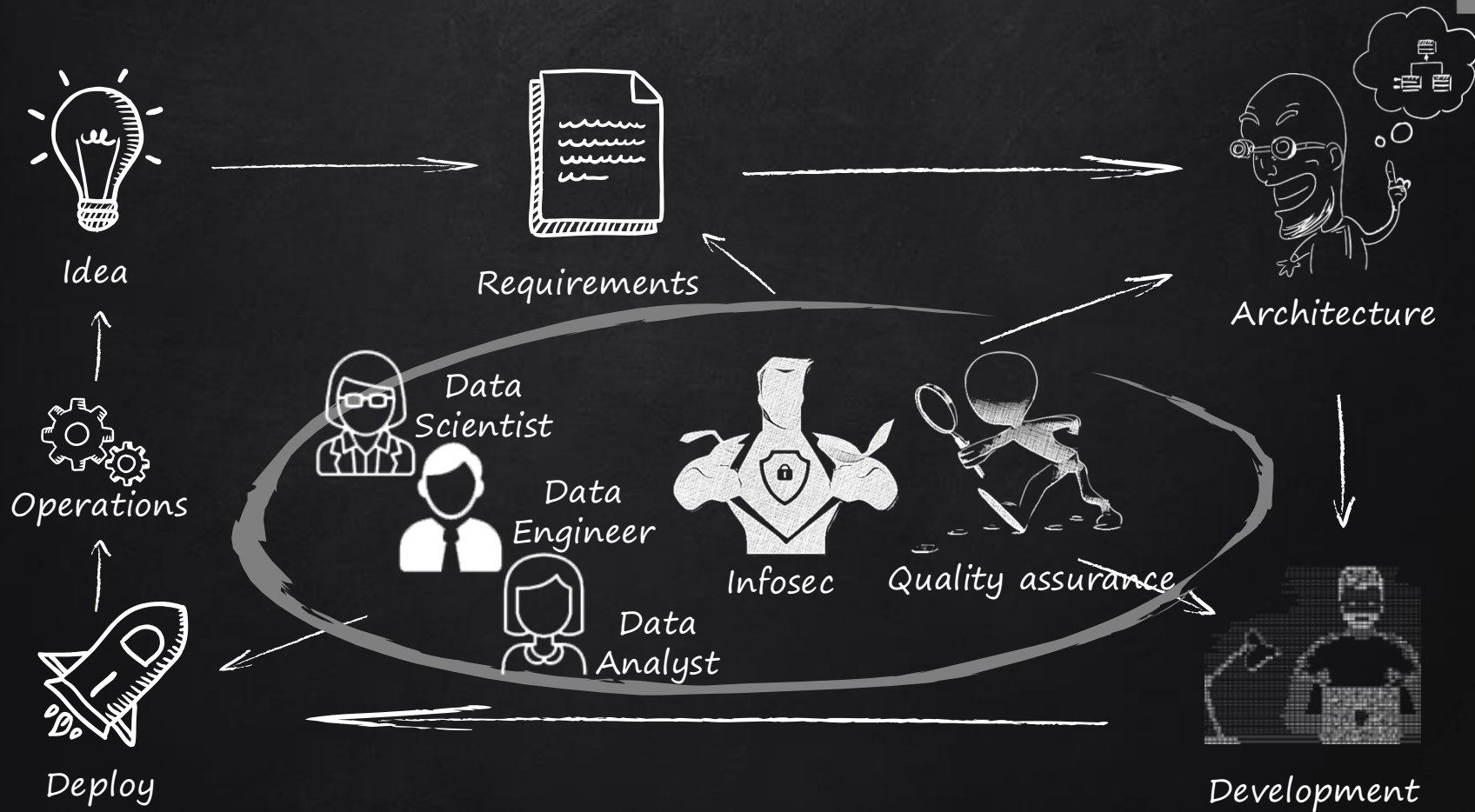
Software delivery performance

Organizational performance

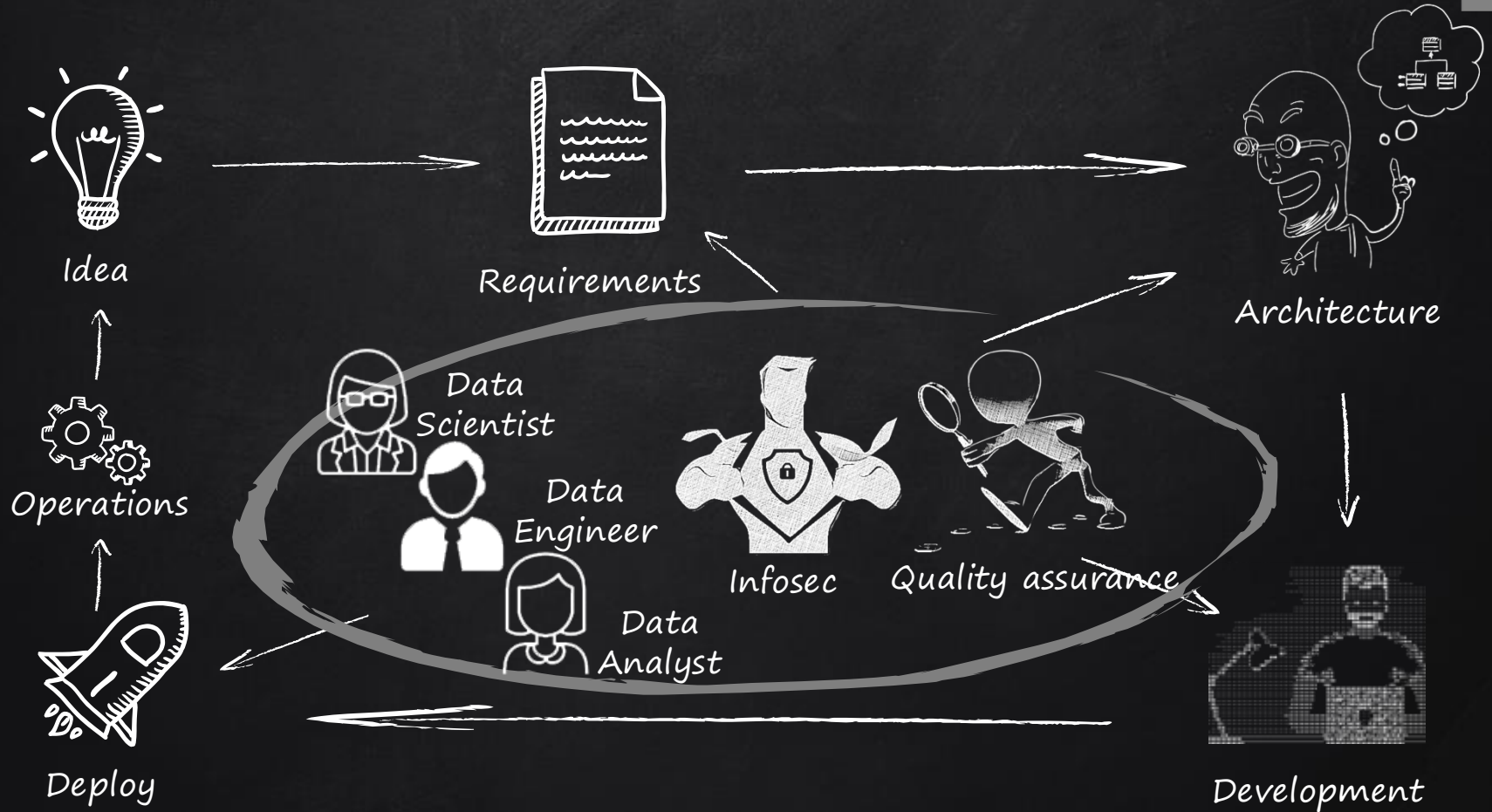
Less rework

Identity

Dev ... Sec ... Data ... WTF ... Ops



Pipeline driven organization



Strategic programming



The first step towards becoming a good software designer is to realize that working code isn't enough. It's not acceptable to introduce unnecessary complexities in order to finish your current task faster...

Your primary goal must be to produce a great design, which also happens to work. This is strategic programming.

John Ousterhout, Raft's co-inventor

More details in the book "A philosophy of software design"

Spirit – our internal:
developer platform



Spirit – our internal developer platform



❖ VCS

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
 - ❖ Managed Kubernetes

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
 - ❖ Managed Kubernetes
 - ❖ DBaaS (Postgres)

Spirit – our internal developer platform



- ❖ VCS

- ❖ XaaS

- ❖ Managed Kubernetes
- ❖ DBaaS (Postgres)
- ❖ Kafka as a Service

Spirit – our internal developer platform



- ❖ VCS

- ❖ XaaS

 - ❖ Managed Kubernetes

 - ❖ DBaaS (Postgres)

 - ❖ Kafka as a Service

 - ❖ Cassandra as a Service

Spirit – our internal developer platform



- ❖ VCS

- ❖ XaaS

 - ❖ Managed Kubernetes

 - ❖ DBaaS (Postgres)

 - ❖ Kafka as a Service

 - ❖ Cassandra as a Service

 - ❖ S3

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
 - ❖ Build infrastructure

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
 - ❖ Build infrastructure
 - ❖ Image registry

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
 - ❖ Build infrastructure
 - ❖ Image registry
 - ❖ Test management system (Allure) + Quality gate

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
 - ❖ Build infrastructure
 - ❖ Image registry
 - ❖ Test management system (Allure) + Quality gate
 - ❖ Performance testing (Cosmos)

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
 - ❖ Build infrastructure
 - ❖ Image registry
 - ❖ Test management system (Allure) + Quality gate
 - ❖ Performance testing (Cosmos)
 - ❖ Fitness functions (Danger)

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
- ❖ Operation

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
- ❖ Operation
 - ❖ Observability platform (Sage) and traces

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
- ❖ Operation
 - ❖ Observability platform (Sage) and traces
 - ❖ SLAser (SLAs for all system)

Spirit – our internal developer platform



- ❖ VCS
- ❖ XaaS
- ❖ Pipelines
- ❖ Operation
 - ❖ Observability platform (Sage) and traces
 - ❖ SLAser (SLAs for all system)
 - ❖ OMG (for incidents)



How to create a culture of reliability?

More details in the white paper “Site Reliability Engineering”

Project Aristotle by Google



What makes a team effective at Google?

Project Aristotle by Google



What makes a team effective at Google?

- ❖ Psychological safety

- Team members feel safe to take risks and be vulnerable in front of each other

Project Aristotle by Google



What makes a team effective at Google?

- ❖ Psychological safety
- ❖ Dependability
 - Team members get things done on time and meet Google's high bar for excellence

Project Aristotle by Google



What makes a team effective at Google?

- ❖ Psychological safety
- ❖ Dependability
- ❖ Structure and clarity
 - Team members have clear roles, plans and goals

Project Aristotle by Google



What makes a team effective at Google?

- ❖ Psychological safety
- ❖ Dependability
- ❖ Structure and clarity
- ❖ Meaning
 - Work is personally important to team members

Project Aristotle by Google



What makes a team effective at Google?

- ❖ Psychological safety
- ❖ Dependability
- ❖ Structure and clarity
- ❖ Meaning
- ❖ Impact
 - Team members think their work matters and creates change

Westrum organizational cultures



Typology of organizational cultures

```
graph TD; A[Typology of organizational cultures] --> B[Pathological (power-oriented)]; A --> C[Bureaucratic (rule-oriented)]; A --> D[Generative (performance-oriented)];
```

Pathological (power-oriented)

Organizations are characterized by large amounts of fear and threat. People often hoard information or withhold it for political reasons, or distort it to make themselves look better.

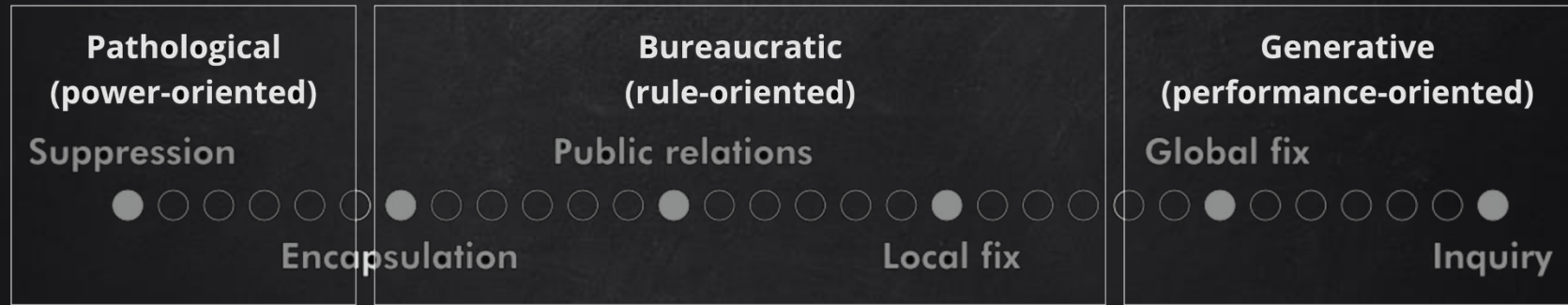
Bureaucratic (rule-oriented)

Organizations protect departments. Those in the department want to maintain their "turf," insist on their own rules, and generally do things by the book—*their* book.

Generative (performance-oriented)

Organizations focus on the mission. How do we accomplish our goal? Everything is subordinated to good performance, to doing what we are supposed to do.

Responses to anomalous information



Suppression—Harming or stopping the person bringing the anomaly to light; “shooting the messenger”

Encapsulation—Isolating the messenger, so that the message is not heard

Public relations—Putting the message “in context” to minimise its impact

Local fix—Responding to the presenting case, but ignoring the possibility of others elsewhere

Global fix—An attempt to respond to the problem wherever it exists. Common in aviation, when a single problem will direct attention to similar ones elsewhere

Inquiry—Attempting to get at the “root causes” of the problem

Postmortems



The primary goals of writing a postmortem are to ensure that the incident is documented, that all contributing root cause(s) are well understood, and, especially, that effective preventive actions are put in place to reduce the likelihood and/or impact of recurrence.

From Google SRE Book

Postmortems



The primary goals of writing a postmortem are to ensure that the incident is documented, that all contributing root cause(s) are well understood, and, especially, that effective preventive actions are put in place to reduce the likelihood and/or impact of recurrence.

From Google SRE Book

I have two public stories about postmortems

Postmortems



The primary goals of writing a postmortem are to ensure that the incident is documented, that all contributing root cause(s) are well understood, and, especially, that effective preventive actions are put in place to reduce the likelihood and/or impact of recurrence.

From Google SRE Book

I have two public stories about postmortems

❖ Culture of postmortems or how we learn from our fuckups

Postmortems



The primary goals of writing a postmortem are to ensure that the incident is documented, that all contributing root cause(s) are well understood, and, especially, that effective preventive actions are put in place to reduce the likelihood and/or impact of recurrence.

From Google SRE Book

I have two public stories about postmortems

- ❖ Culture of postmortems or how we learn from our fuckups
- ❖ From monolith to microservices and back again at Fuckup Nights



@BOOK_CUBE

t.me/book_cube/1575