

Set of **techniques and technologies**

applied in a

Google Cloud Platform environment

to maintain **multiple** geospatial oriented **projects** keeping an eye on
sustainability, maintainability, reliability, limitations and ... “best practices” :)

Carlo Cancellieri

Environment

Actors (in order)

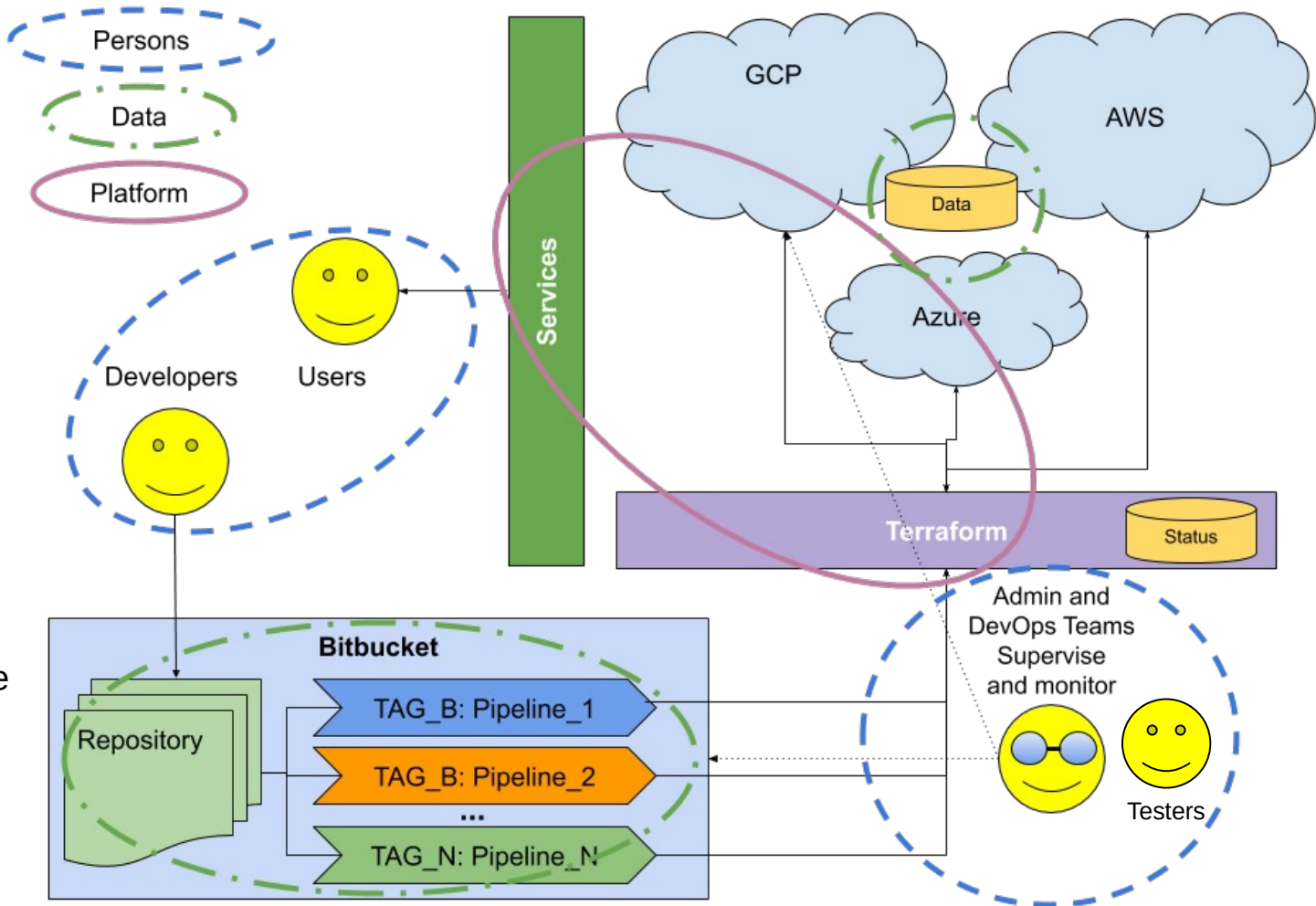
1. **Persons**
2. **Data**
3. **Platform**

Integrations

Integration is the key to success. Well defined, **standard layers**, **interfaces** and **FEW CLEAR conventions**

Tracking changes

Changes might be **documented** to share the knowledge of the status of the platform



Users



Open

Most of the services are **open to read** and **write** (for authenticated users)

We're **providing accounts automatically/on demand***

Changes are **usable** in stories **in few minutes**

Easy

All the services are **standard** and **documented by the community**

It's easy to use well known tools like **Qgis**, **ArcGis** or **GEE***

Integrated

We are **not forcing users** to work with specific tools

we are *working to **integrate** more **Open Source tools** and protocols (f.e. WPS)

but also to provide integrations with **GEE** and much more*

Accessible, stable and Integrated

Team / DevOps



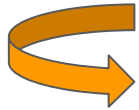
**Development might not affect the DevOps team activities
for this reason (and many more)
environments (gcp projects) are well separated**

Activities over **Review** are **safe** (by design)

No need to ask for delivery

[but let's **keep** DevOps **informed**]

It's easy to **deploy** and **test autonomously**



[**deploy from scratch**

then

destroy all to save resources]

Activities over **Production** must be **supervised**

Dedicated projects (or branches)

MUST be **authorized** and **performed** in a **PAIR SESSION** with a **DevOps** responsible

[to **share the knowledge** and avoid mistakes]



Pair sessions, knowledge sharing

Team / Testers

Testers



Unidirectional

- New **features** are **delivered in Review**, **tested** then, **ported*** to **Production**



Test driven

- Every* time a migration from Review to Production is performed a **full coverage set of tests is performed over the services** (Jmeter, other, ...)

Continuous Testing



- Pipelines* for **continuous testing**

Fail fast, F.i.r.s.t. principles: Fast, Isolated/Independent, Repeatable, Self-validating, Timely

Team / Developers



Developers

**Independent release cycles for
the Solution(s)
and
the Product(s)**

**Well* defined and extensible* interface
between Product(s) and IAC project (Solution)**

Continuous Integration pipelines

Decoupled, Isolated, Maintainable

Data protection

Unidirectional flow from **Review** to **Production**,
from **Primary** (master) to **Secondary** instances (slaves)



Experiments are performed over a copy in **Review** then **Replicated** in **Production**
[once the procedure is clear]

Automatic **scheduled backup** up to 14 days [**Disks** and **Database**]



 **Rollback ready:** it's easy [in **emergency**] to
create a temporary machine **serving data** from a previous **backup**

Data: Unidirectional flow, clear workflow, versioning, fast recovery

Platform

Immutable - Reproducible

Using tags any (infrastructure) release is reproducible

Modular

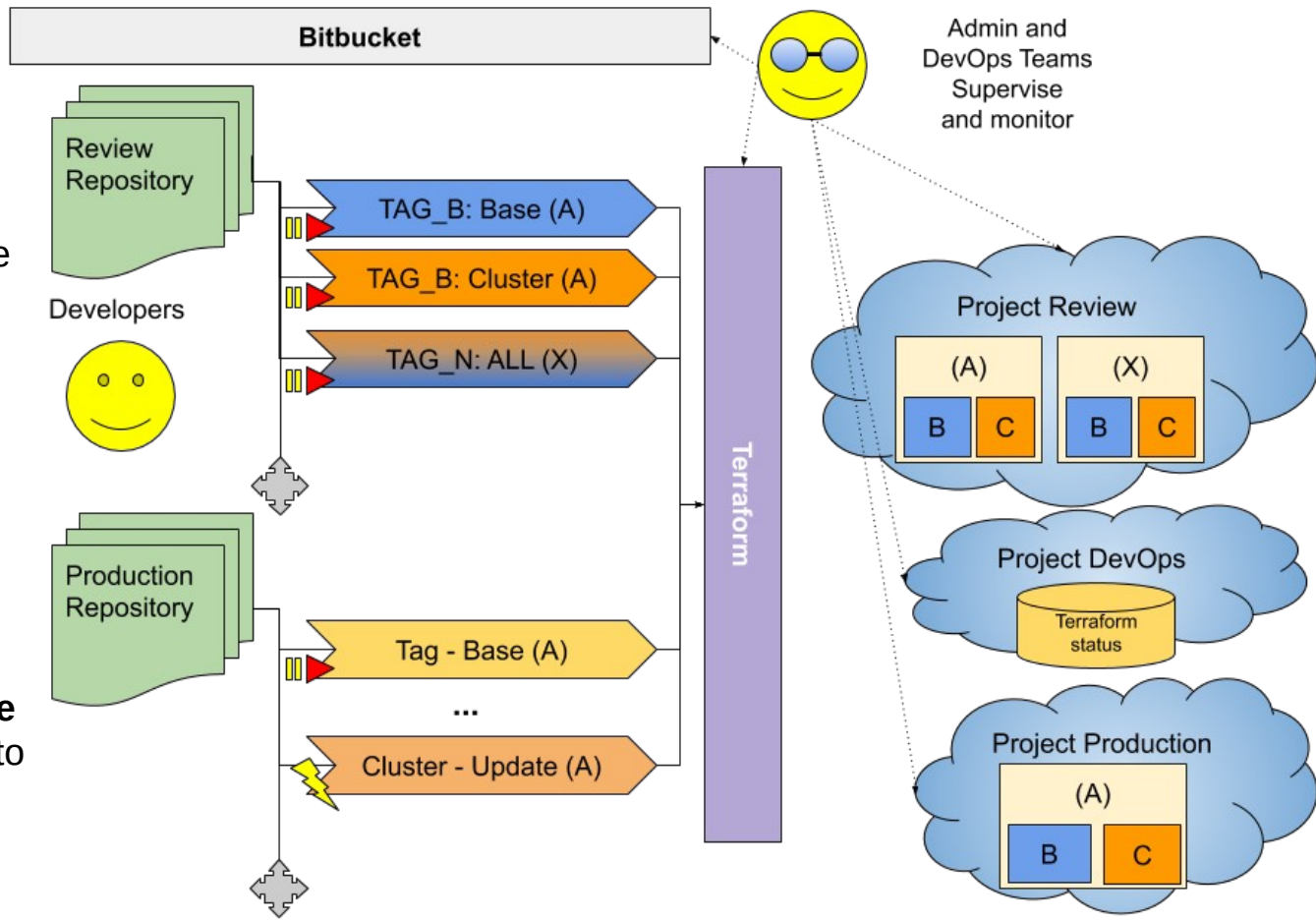
All the (Terraform) **modules** are **intentionally generic**, it's simple to **reuse** them

Maintainable

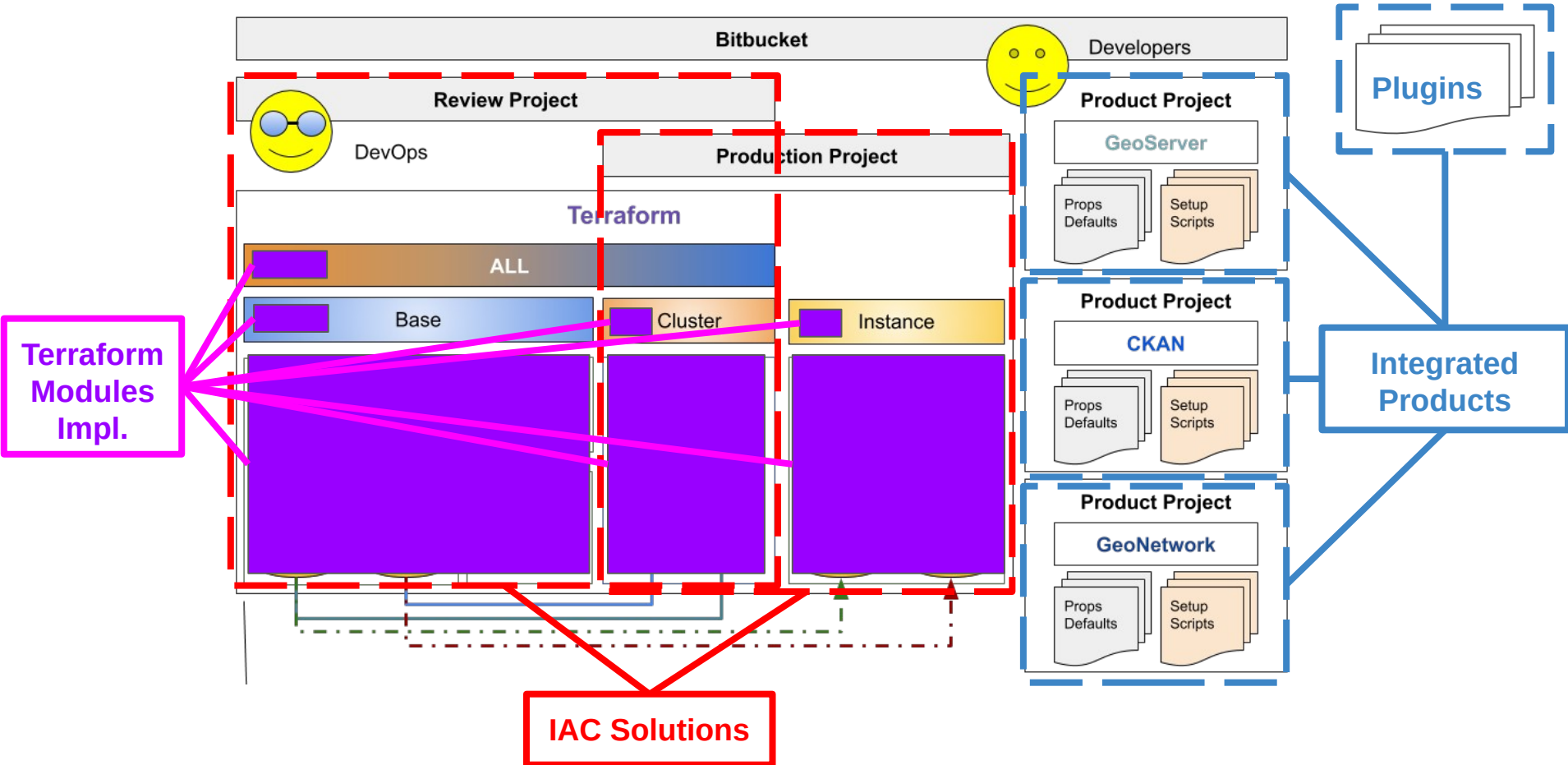
Easy to create/**upgrade** thanks to the **automatic** propagation. Destroy will **easily take care of cleanup** (in Review).

Scalable

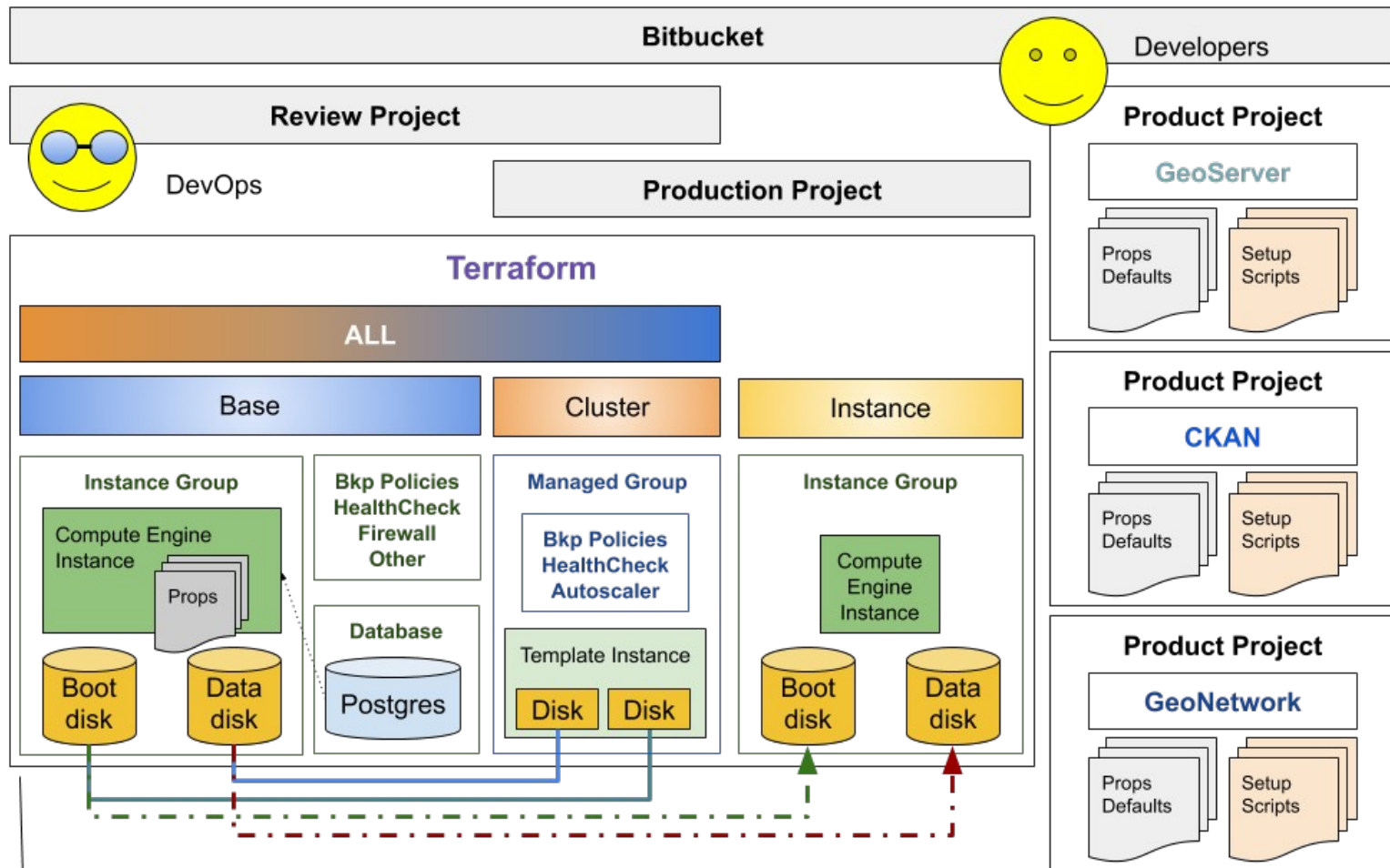
Designed to **automatically scale up and down** to N+1 instances to support up to thousands of requests per minute.



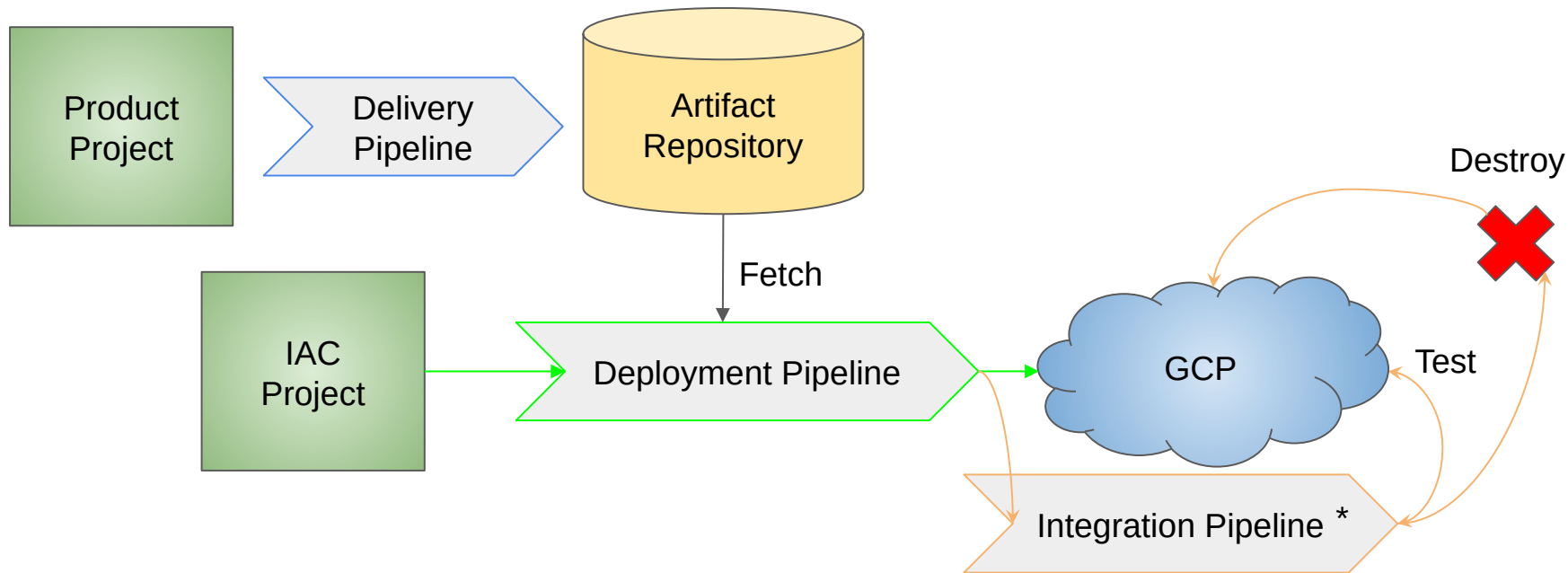
Modularization and packaging



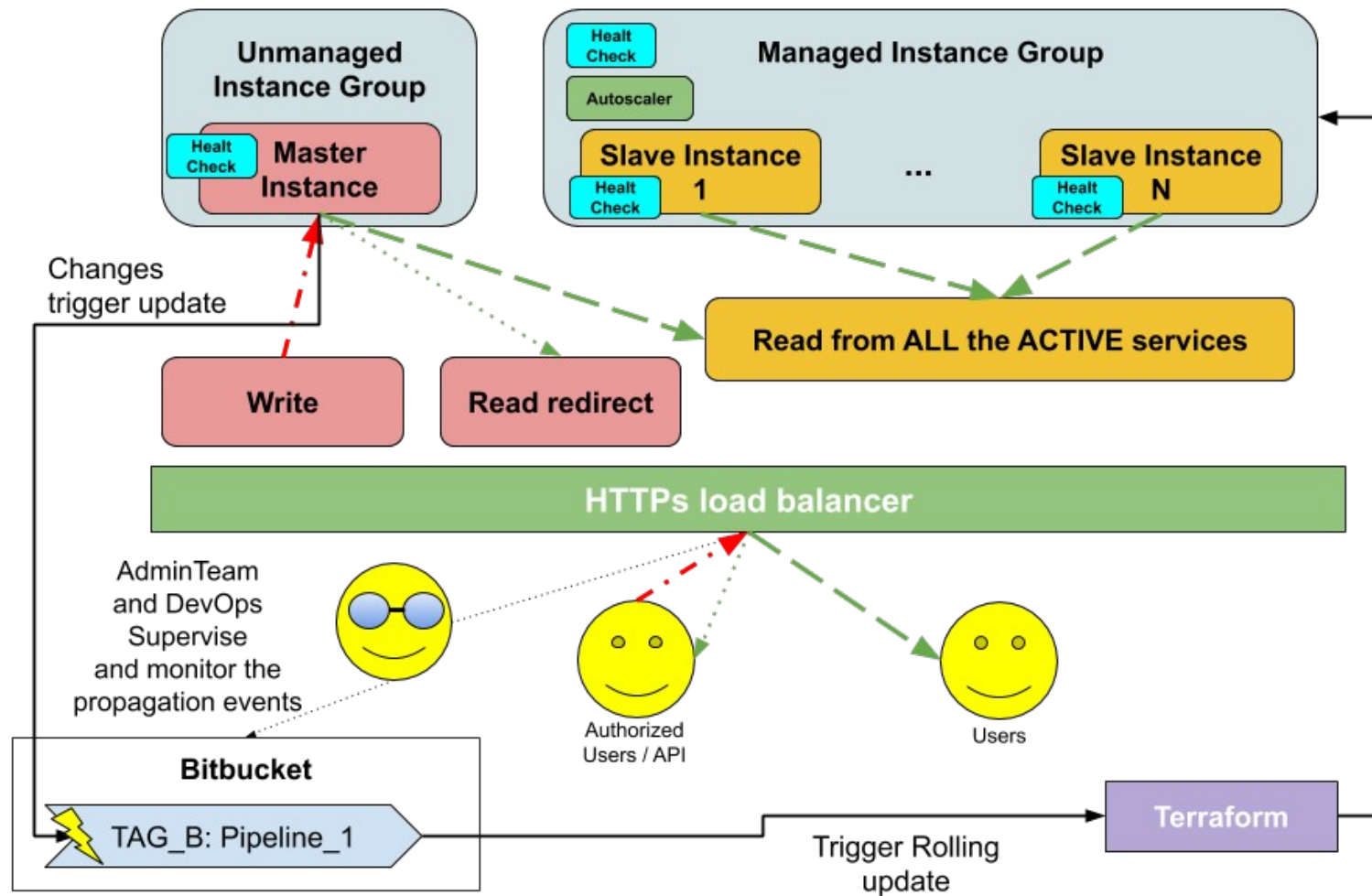
Products and Solutions



Delivery, Deploy, Test Continuously



Master - Slave | Active - Active



Master - Slave | Active - Active

Master

Managed Instance Group

Load Balancer

Changes trigger u

Admin Team and DevOps Supervise and monitor the propagation events

Bitbucket

TAG_B: Pipeline_1

Authorized Users / API

Trigger Rolling update

Terraform

Unity (US, JP)

FASTER, (US, JP, TW)

PLCN (US, TW)

Curie (CL, US)

Submarine Cable Investments

Current Network

SELECT TO VIEW:

Edge Point of Presence

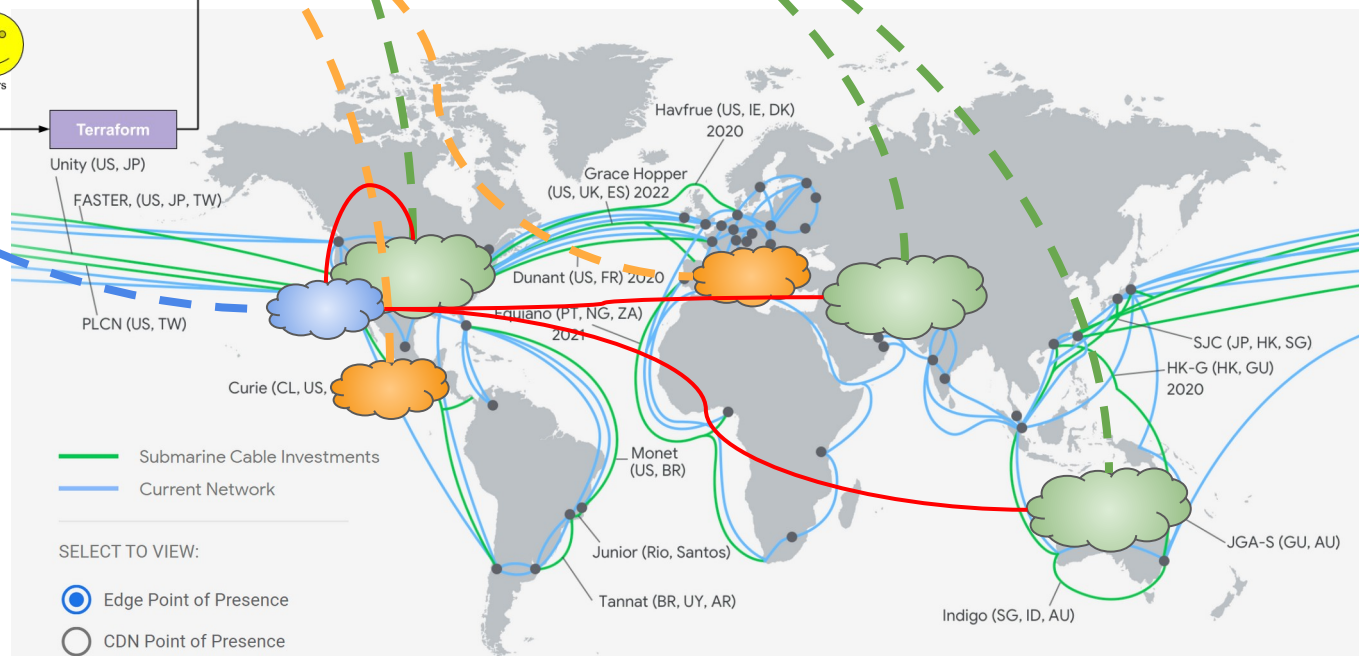
CDN Point of Presence

Why propagation:

1. Cloud Ready
2. Reduce **latency**
3. Improve **availability**
4. Better to have a **Replica**

Google Cloud Inter Region Latency

Google Cloud supports 18 regions for its cloud services. Understanding latency between the regions is important as performance starts to have noticeable degradation when the latency is more than 150 - 200ms between a client and server or between two servers in two different regions.



Keeping secondary groups up to date (over regions)

Operational screenshots omitted from public copy

Upload pipeline is **triggered** from the **master** instance **each 30 minutes** (if a **configuration or data change** is detected).

Thanks to **Terraform** it provides:

- new **Instance template**
- attaching **fresh disks copy** and policies
- **triggers** the **rolling update** on the **managed instance group**
- outdated disks and templates cleanup



Questions?

Want more?



Developers

The following slides will clarify some important details over the internal mechanisms of the Product projects

Internal documentation is provided inline in the code

Product Projects:

Each **release** is made to **configure a Debian/Ubuntu vanilla image from scratch** providing a **fully working and connected** (to **disks** and **database**) **running instance** of:

- [Debian] **GeoServer**
- [Ubuntu] **CKAN**
- [Debian] **GeoNetwork**

CI/CD Pipelines:

Each **product project** produces **2 artifacts**:

- **setup.sh**
- **resources.tar.gz**

These artifacts are deployed at each commit over the \$RELEASE google bucket

The release **also provides some overrides** for the **incoming customization variables (Terraform)** such as:

- **Context path** to expose the web service (/path/to/context)
- List of **plugins to install** (several)
- Ancillary files to **configure plugins** (several)
- Other **optimizations** (JVM, Connection pools, memory usage, etc)

Compute Instance:

OnShutdown:

- **[log_process.sh]** Collect logs and backup in a tarball over a bucket
- **[audit_process.sh]** Backup all the pending monitoring (requests) logs over a bucket

Daily:

- **[audit_process.sh]** Backup all the incoming requests logs over a bucket

OnStartup:

- **[startup-script.sh]** Run:
 - Source environment variables
 - download fresh **Setup.sh** from the **\$RELEASE** bucket
 - Run **Setup.sh**

On [Setup.sh]:

- Download fresh **resources.tar.gz**
- If lock files are present, no action is performed
- Lock files to prevent further execution on boot
- Install system software checking for errors
- If requested initialize the **disk** and add to mount
- If requested initialize **DB** and change password
- Update/install **GeoServer/CKAN/GeoNetwork**

Main (Master) instance:

Each half hour:

- **[run_tag.sh]** Run the smart script to check for changes.
If an **update is found** a **pipeline trigger event is fired** to propagate changes.

Secondary (Slave) instance:

OnStartup:

- **[startup-script.sh]** Run:
 - initialize environment variables
 - download fresh **Setup.sh** from the **\$RELEASE** bucket
 - Run **Setup.sh** [normally no action is performed if lock files are present]
 - Initialize crontab to removing **run_tag.sh** execution to avoid duplicate execution

Thank you

Carlo Cancellieri