Cloud Computing

> The entire history of software engineering is that of the rise in levels of abstraction.

-- Grady Booch

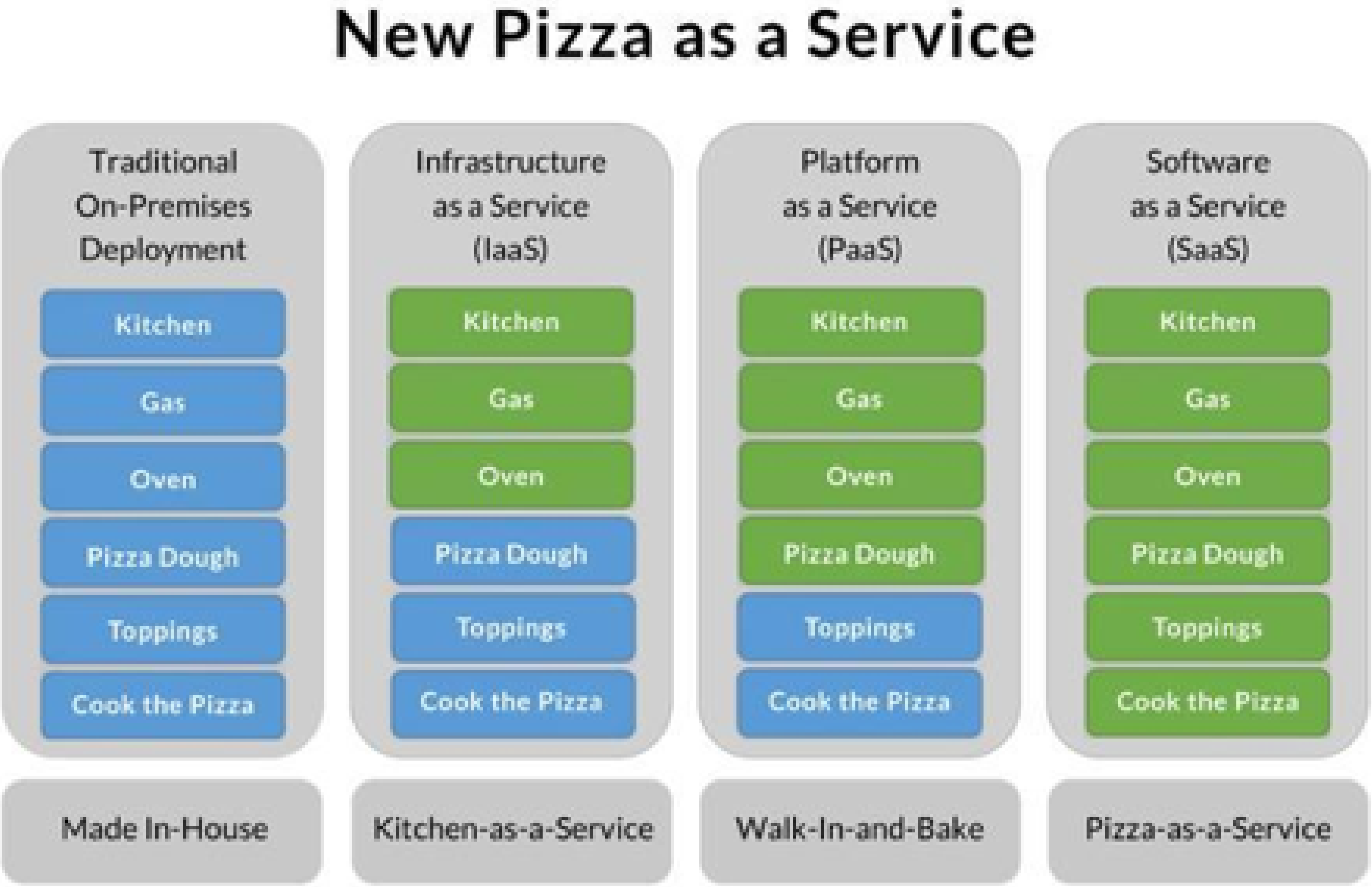
> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

https://csrc.nist.gov/pubs/sp/800/145/final

Essential Characteristics:

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

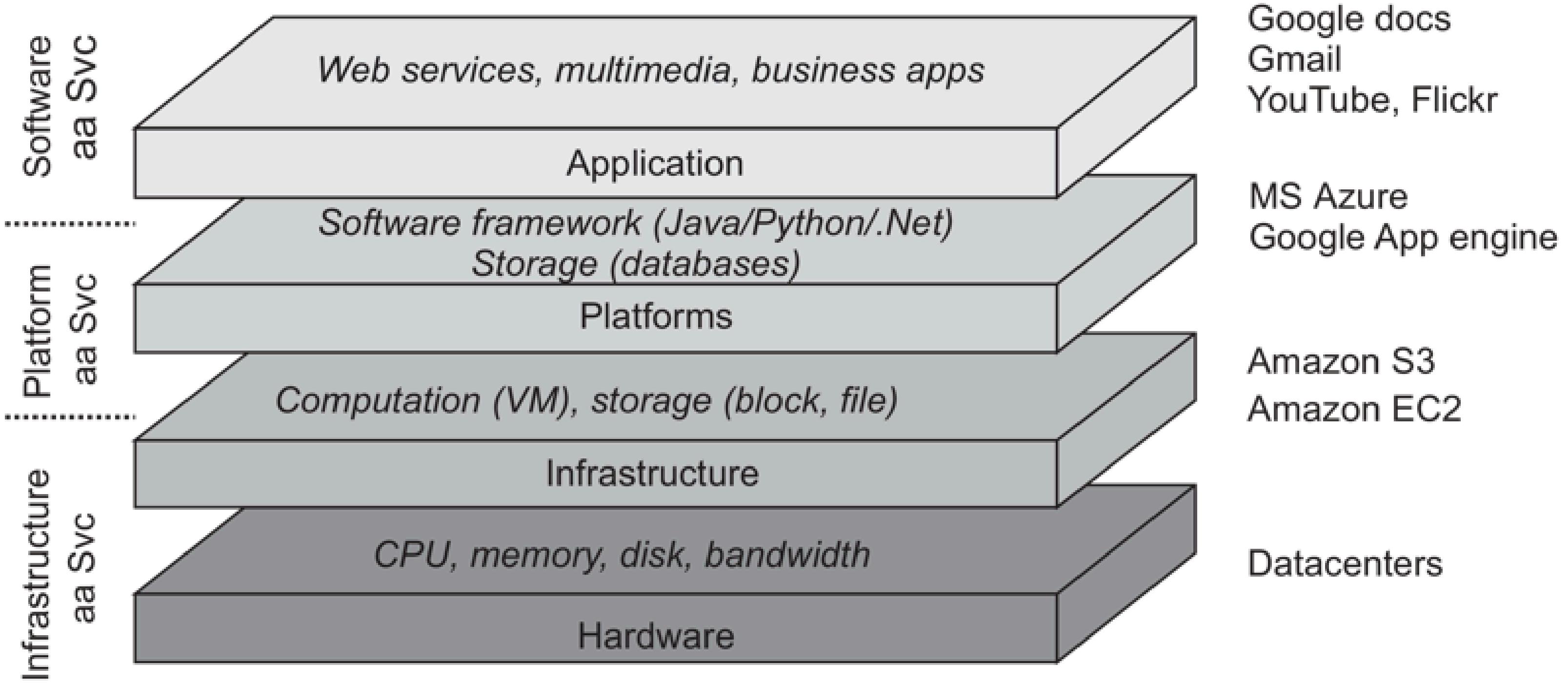
Service Models



New Pizza as a Service

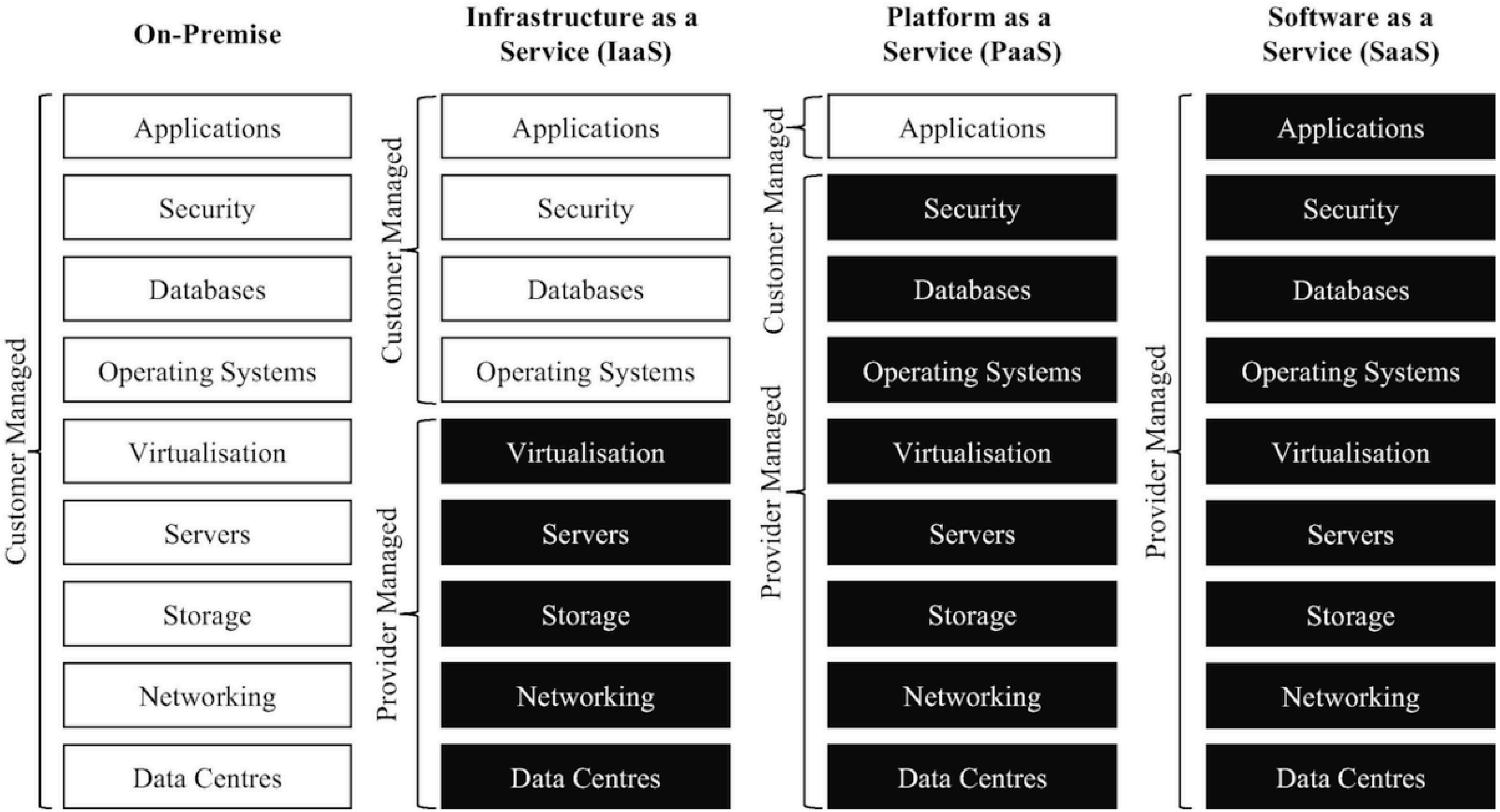
Traditional On-Premises Deployment	Infrastructure as a Service (IaaS)	Platform as a Service (PaaS)	Software as a Service (SaaS)
🟦 Kitchen	🟩 Kitchen	🟩 Kitchen	🟩 Kitchen
🟦 Gas	🟩 Gas	🟩 Gas	🟩 Gas
🟦 Oven	🟩 Oven	🟩 Oven	🟩 Oven
🟦 Pizza Dough	🟦 Pizza Dough	🟩 Pizza Dough	🟩 Pizza Dough
🟦 Toppings	🟦 Toppings	🟦 Toppings	🟩 Toppings
🟦 Cook the Pizza	🟦 Cook the Pizza	🟦 Cook the Pizza	🟩 Cook the Pizza
Made In-House	Kitchen-as-a-Service	Walk-In-and-Bake	Pizza-as-a-Service

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)



(VanSteen, 2017, S. 30)

XaaS



	On-Premise	Infrastructure as a Service (IaaS)	Platform as a Service (PaaS)	Software as a Service (SaaS)
Applications	Customer Managed	Customer Managed	Customer Managed	Provider Managed
Security	Customer Managed	Customer Managed	Provider Managed	Provider Managed
Databases	Customer Managed	Customer Managed	Provider Managed	Provider Managed
Operating Systems	Customer Managed	Customer Managed	Provider Managed	Provider Managed
Virtualisation	Customer Managed	Provider Managed	Provider Managed	Provider Managed
Servers	Customer Managed	Provider Managed	Provider Managed	Provider Managed
Storage	Customer Managed	Provider Managed	Provider Managed	Provider Managed
Networking	Customer Managed	Provider Managed	Provider Managed	Provider Managed
Data Centres	Customer Managed	Provider Managed	Provider Managed	Provider Managed

```
Low Cost                                           High Cost
==============================================================
FARM     WHOLESALER     GROCERY     RESTAURANT     DOORDASH
BUILD    CO-LOCATION    HETZNER     AWS            VERCEL
```

https://news.ycombinator.com/item?id=45614922#45616049

Deployment Models

- Private cloud
- Community cloud
- Public cloud
- Hybrid cloud

Cloud Native Applications

- A "cloud native" application has adapted and evolved to be maximally efficient in its environment: the cloud.
- In the cloud, an application becomes **distributed**.
- It is forced to be **resilient to** hardware/network unpredictability and **unreliability**.

(vgl. https://www.reactiveprinciples.org/cloud-native/index.html)

- Ensuring responsiveness and reliability in this environment is difficult.
- The applications we build after embracing this environment better match how the real world actually works.
- This provides **better experiences** for our users, whether humans or software.

(vgl. https://www.reactiveprinciples.org/cloud-native/index.html)

The constraints of the cloud environment include:

- All inter-service communication takes place over unreliable **networks**.
- You must operate under the assumption that the underlying **hardware can fail** or be restarted or moved at any time.
- The services need to be able to **detect and manage failure** of their peers—including partial failures.
- Strong consistency and transactions are expensive. Because of the coordination required, it is difficult to make services that manage data available, performant, and scalable.

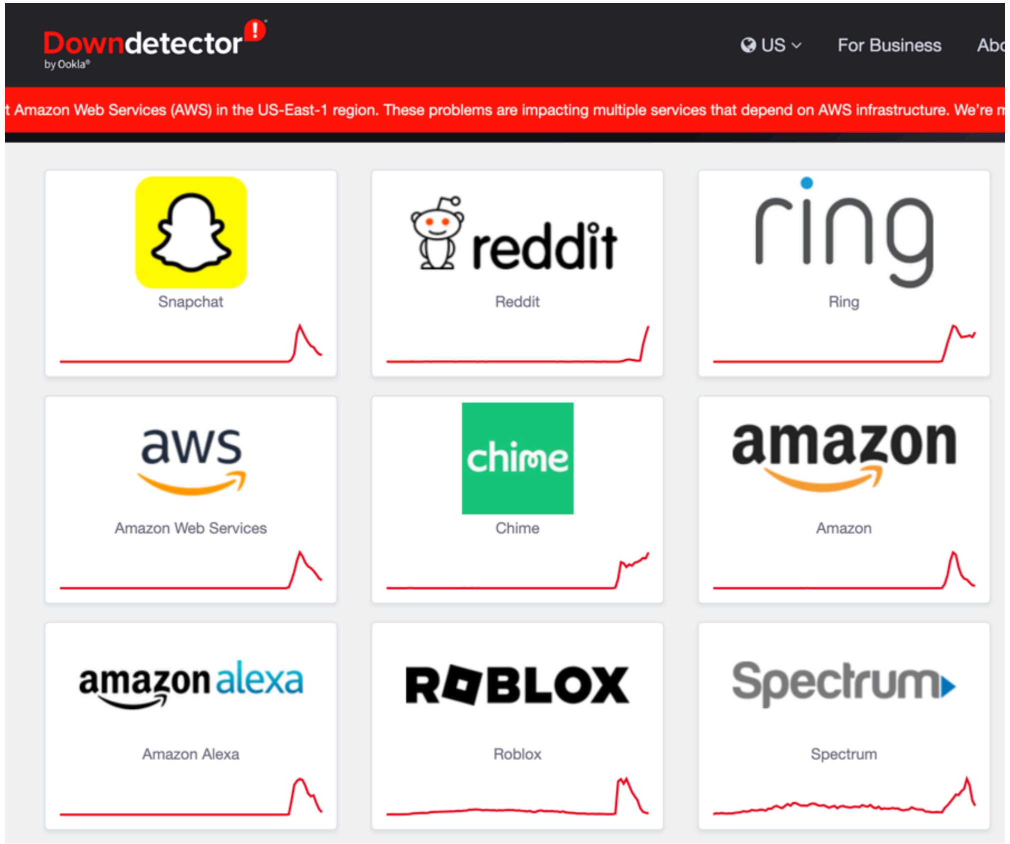
(vgl. https://www.reactiveprinciples.org/cloud-native/index.html)

Therefore, a Cloud Native application is designed to leverage the cloud operating model.

It is predictable, decoupled from the infrastructure, right-sized for capacity, and enables **tight collaboration between development and operations**.

It can be decomposed into loosely-coupled, independently-operating services that are resilient from failures, driven by data, and operate intelligently across geographic regions.

Chancen und Risiken



t Amazon Web Services (AWS) in the US-East-1 region. These problems are impacting multiple services that depend on AWS infrastructure. We're m

Kostenkontrolle

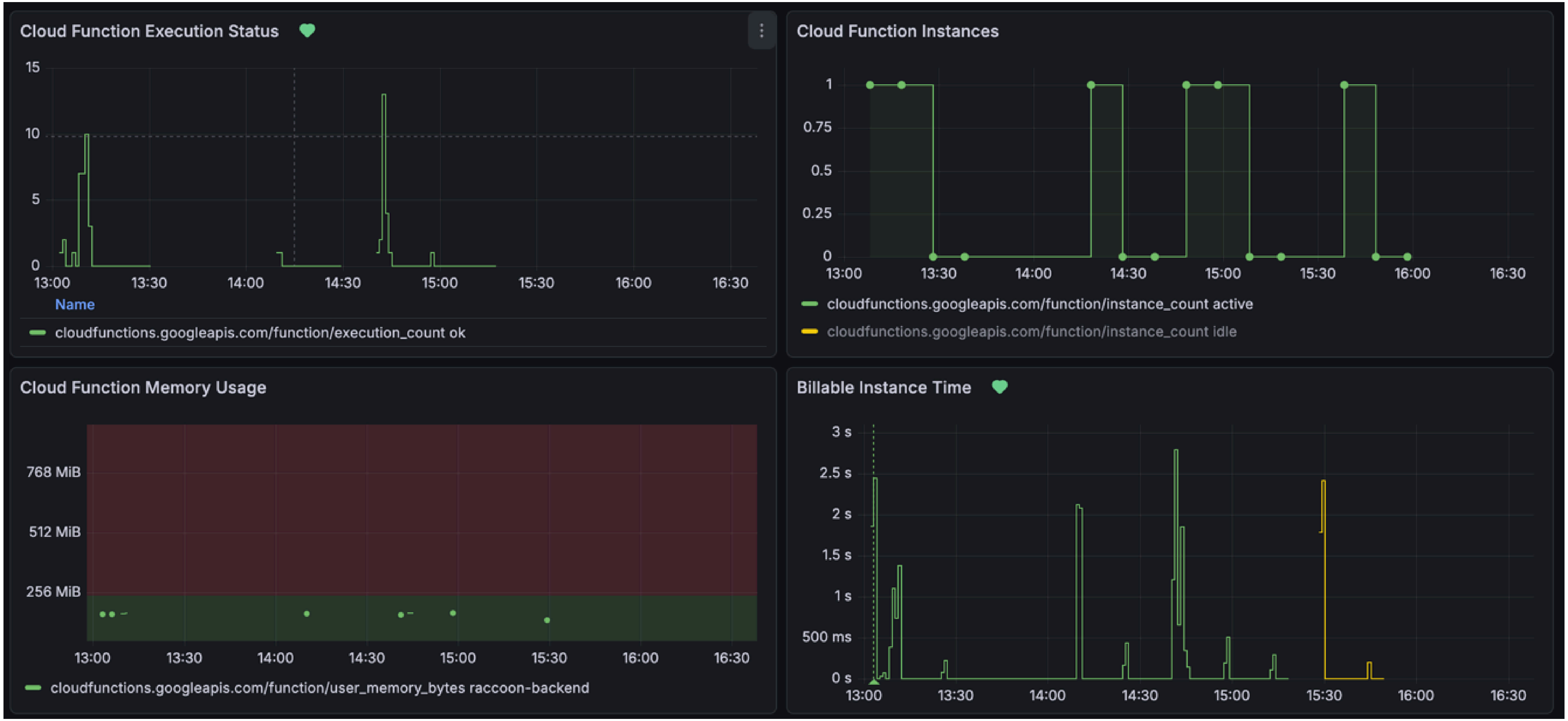
- Serverless Systeme **skalieren** per Definition **automatisch** (Emison S.26)
- Die **Kosten skalieren** dabei mit den Ressourcen mit
- Das kann zu sehr **hohen Kosten** führen (Programmierfehler, Slashdotting, DDoS)
- Deshalb müssen Ressourcen **limitiert und überwacht** werden

Limitierung

```
resource "google_cloudfunctions2_function" "backend" {
  name        = var.backend_function_name
  ...
  build_config {
    runtime     = "nodejs22"
    entry_point = local.function_entry_point
    source { ... }
  }

  service_config {
    max_instance_count               = 2
    max_instance_request_concurrency = 80
    available_memory                 = "256M"
    available_cpu                    = "1"
    timeout_seconds                  = 60
  }
}
```

Monitoring


Cloud Function Execution Status
15
10
5
0
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
Name
cloudfunctions.googleapis.com/function/execution_count ok
Cloud Function Instances
1
0.75
0.5
0.25
0
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
cloudfunctions.googleapis.com/function/instance_count active
cloudfunctions.googleapis.com/function/instance_count idle
Cloud Function Memory Usage
768 MiB
512 MiB
256 MiB
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
cloudfunctions.googleapis.com/function/user_memory_bytes raccoon-backend
Billable Instance Time
3 s
2.5 s
2 s
1.5 s
1 s
500 ms
0 s
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30

Zugriffskontrolle

- Wenn Cloud-Access-Keys in falsche Hände geraten, kann dies zu **hohen Kosten, Datendiebstahl oder Ausfällen** führen
- Secrets werden am Besten in **Secret Managern** gespeichert
- Secrets sollten niemals in Git eingecheckt werden. Dies kann mit Scannern **verhindert oder detektiert** werden
  - https://thoughtworks.github.io/talisman/
  - https://trivy.dev/
  - Wenn Secrets dennoch eingecheckt werden, müssen sie **sofort geändert** werden (Rotation), am besten automatisiert.
- Access-Keys sollten nur über die **minimal nötigen Rechte** verfügen (https://en.wikipedia.org/wiki/Principle_of_least_privilege)

```
gcloud projects add-iam-policy-binding $PROJECT_ID \
  --member="serviceAccount:terraform@example.iam.gserviceaccount.com" \
  --role="roles/storage.admin" \
  --role="roles/cloudfunctions.admin" \
  --role="roles/iam.serviceAccountUser" \
  --role="roles/iam.serviceAccountAdmin"
```

Quellen

Emison : Joseph Emison (2024): Serverless as a Game Changer, Pearson