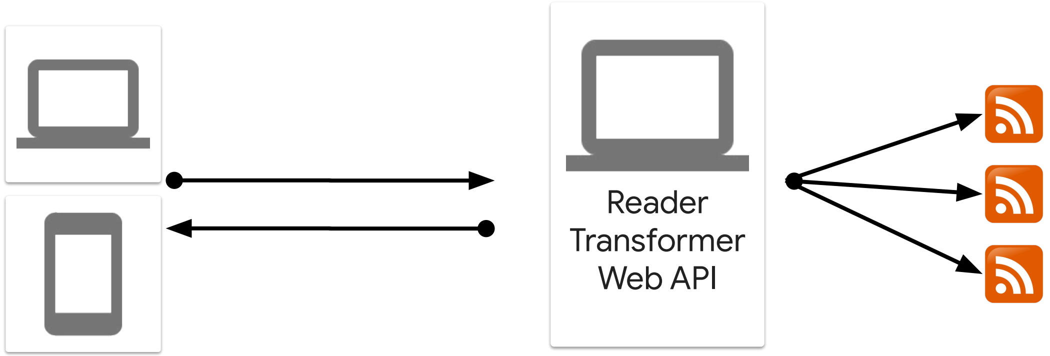
An app modernization story with Cloud Run

Mete Atamel
Developer Advocate at Google
@meteatamel
atamel.dev/tags/app-modernization

Stage 0: Prototype (Late 2015 / Early 2016)

Goal: Get something up and running



Server: ASP.NET (4.6) Windows app on IIS hosting

Client: Android and iOS app with Ionic Framework

Prototype: Pros & Cons

+Worked!

+Easy to understand

+Easy to deploy

+Inexpensive

-Too much coupling

-Bad DevEx (FTP to see logs!)

-No redundancy

-No persistence

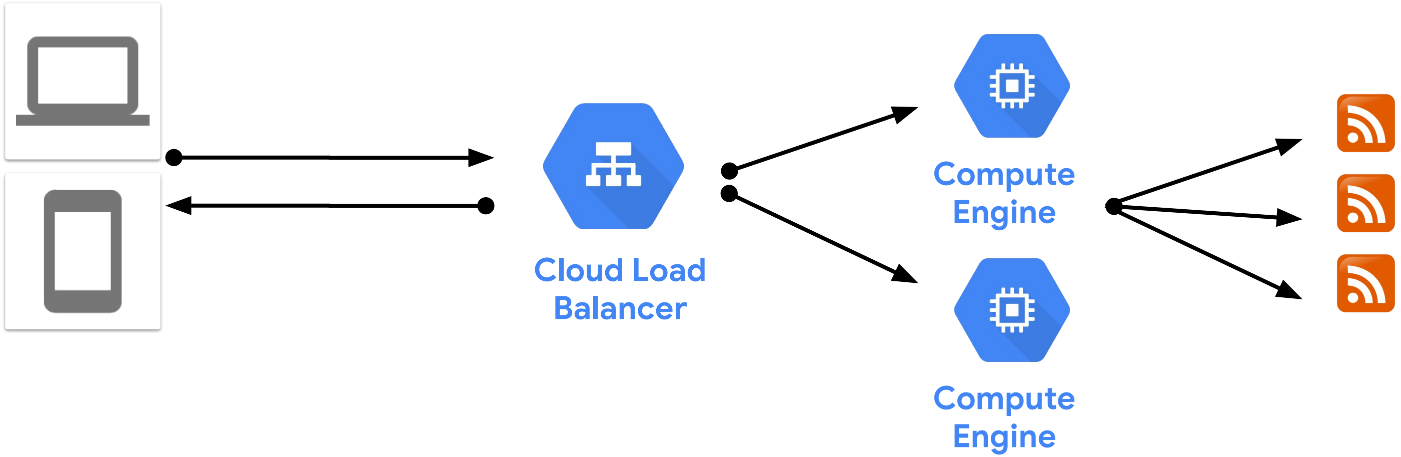
-No resilience

Prototype: Lessons Learned

1. Stick to MVP
2. Research your options
3. Avoid coupling at all costs
4. Design with future in mind

Stage 1: Lift & Shift (Late 2016 / Early 2017)

Goal: Improve resiliency and redundancy



Compute Engine Windows VMs on Google Cloud

Lift & Shift: Pros & Cons

+Easy to move with ASP.NET Framework Template

+Redundancy & load-balancing with Instance Template & Groups

+Possiblity of autoscaling

+Much better DevEx with Stackdriver logging, VM snapshots etc.

-More expensive than IIS hosting

Lift & Shift: Lessons Learned

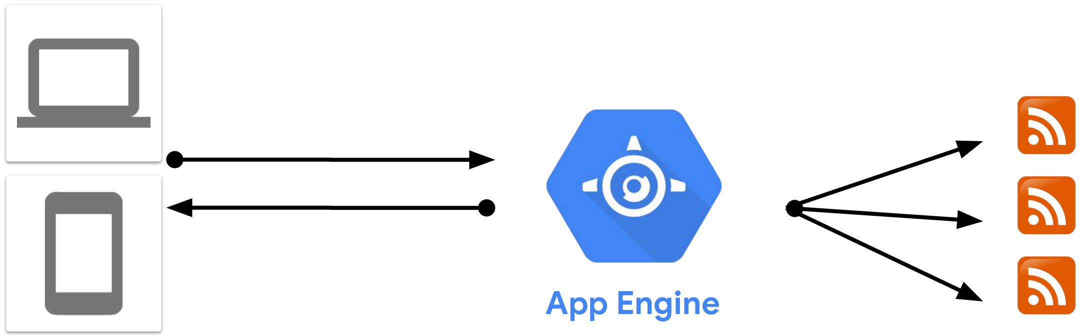
1. Moving to Cloud was not that difficult
2. Cloud is much more than just hosting

The app served us well until 2019…

1. .NET Core
2. Windows dependency
3. Containers
4. Costs

Stage 2: Containarization (Early 2019)

Goal: Remove Windows dependency and cost



Re-write in ASP.NET Core (2.2), containerize w/ Docker & deploy to App Engine Flex (Linux)

Containarization: Pros & Cons

+Windows license fees out

+Free autoscaling

+Revision management

+Traffic splitting

-VM based

-Pricing

-Slow deploys

Containarization: Lessons Learned

1. Refactor for clear benefits
2. Solid functional tests are crucial
3. Project organization matters
4. There’s no magic bullet

Cloud Run

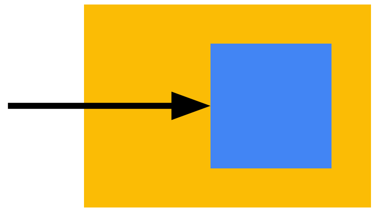
Bringing serverless to containers

Container to production in seconds

Natively Serverless

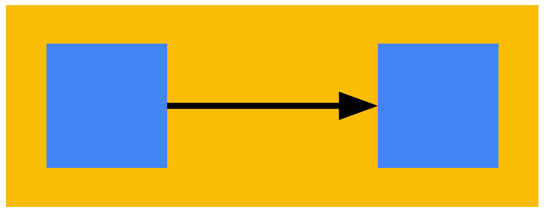
One experience, where you want it

HTTPS Endpoint



Public

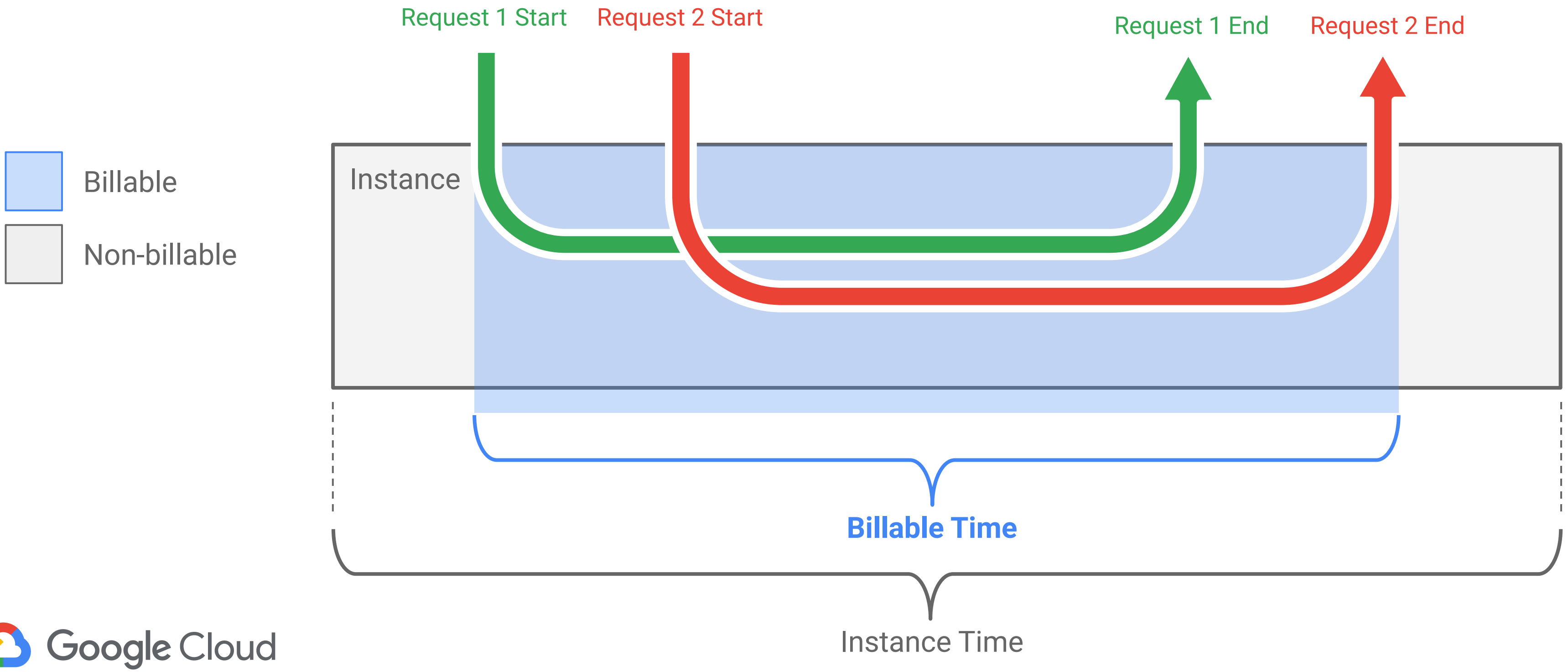
- Website
- API endpoint
- Mobile backend
- Webhook



Private

- Internal services
- Async tasks

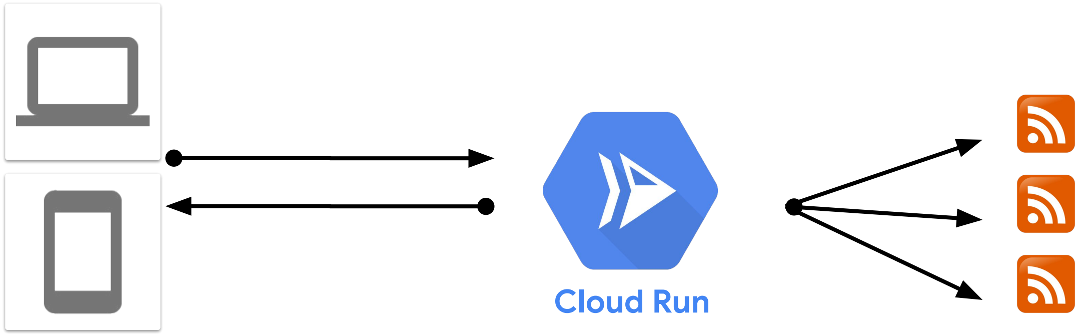
Billable time



Google Cloud

Stage 3: Serverless (Mid 2019)

Goal: Move from VM minute-based pricing to serverless pricing



Update to ASP.NET Core (3.0) & deploy to Cloud Run

Serverless: Pros & Cons

+No VMs

+Serverless billing, much cheaper

+Quick deployments (seconds)

+Great DevEx (integrated logging, revision and traffic management, etc.)

+Based on open-source Knative

-Still a monolith with monolith issues

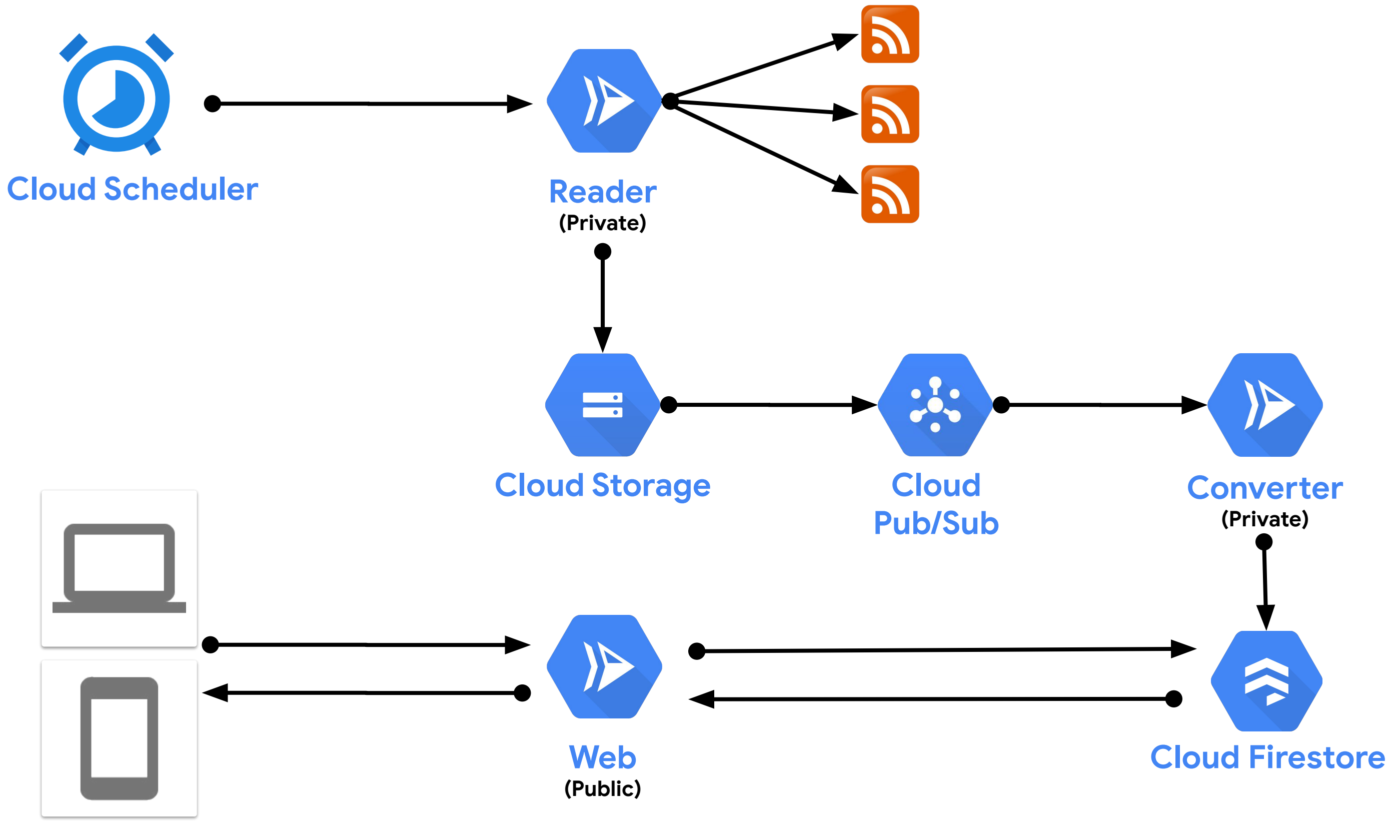
Monolith issues

1. Scaling: all or nothing
2. Cold starts
3. In-memory state
4. No way to update individual services

Monolith decomposition questions

1. How do you break the monolith?
2. How do microservices communicate?
3. How do you handle persistence without coupling?

Stage 4: Monolith to microservices (Early 2020)



Monolith to microservices: Pros & Cons

+Loosely coupled architecture

+Ability to update individual pieces

+Ability to use different languages

+Better utilization of resources

+Persistence

-Many moving parts

-More complex deployment

-Probably more expensive than a monolith?

Grand Lessons Learned

- Transformation does not have to be all or nothing
- Even simple lift & shift can have huge benefits
- Non-optimal solutions can be a stepping stone to more optimal solutions
- Expect some kind of re-write for cloud at some point
- Monolith decomposition is hard! Need a good reason beyond separation of concerns

Thank you!

@meteatamel
atamel.dev/tags/app-modernization

github.com/meteatamel/amathus

github.com/meteatamel/cloudrun-tutorial
cloud.google.com/run