# Distributed Systems: Google Cloud

Kristof Jannes

17 October 2023

KU LEUVEN DistriNet

# Recap lab sessions

- Remote communication: completed
  - **Deliverables: report (20 Oct)**
- Distributed cloud applications
  - Mandatory Level 1
  - Optional Level 2

# Google Cloud Platform

# Intro GCP

Google Cloud Platform

Competitors:

ORACLE Cloud Infrastructure

IBM Cloud

aws

Alibaba Cloud

# Intro GCP: Compute

Google Cloud Platform

| Compute Engine | Container Engine | App Engine |
|---|---|---|
| Application | Application | Application |
| Runtime | Runtime | Runtime |
| Guest OS | Container | Sandbox |
| Hypervisor | Container Engine | Sandbox Engine |
| OS | OS | OS |
| Hardware | Hardware | Hardware |
| IaaS | IaaS | PaaS |

Provider managed  User managed

# Intro GCP: Other services

Google Cloud Platform

## Networking

- Cloud Virtual Network
- Cloud Load Balancing
- Cloud CDN
- Cloud Interconnect
- Cloud DNS

## Big Data

- BigQuery
- Cloud Dataflow
- Cloud Dataproc
- Cloud Datalab
- Cloud Pub/Sub
- Genomics

## Networking

- Cloud Virtual Network
- Cloud Load Balancing
- Cloud CDN
- Cloud Interconnect
- Cloud DNS

## Machine Learning

- Cloud Machine Learning
- Vision API
- Speech API
- Natural Language API
- Translation API
- Jobs API

## Storage and Databases

- Cloud Storage
- Cloud Bigtable
- Cloud Datastore
- Cloud SQL
- Persistent Disk

## Identity & Security

- Cloud IAM
- Cloud Resource Manager
- Cloud Security Scanner
- Cloud Platform Security

...

# Topics for today

- Google App Engine
- Firebase Authentication
- Cloud Pub/Sub
- Cloud Firestore
- Spring Boot
- Lab sessions: Cloud

# Google App Engine

# What is Google App Engine?

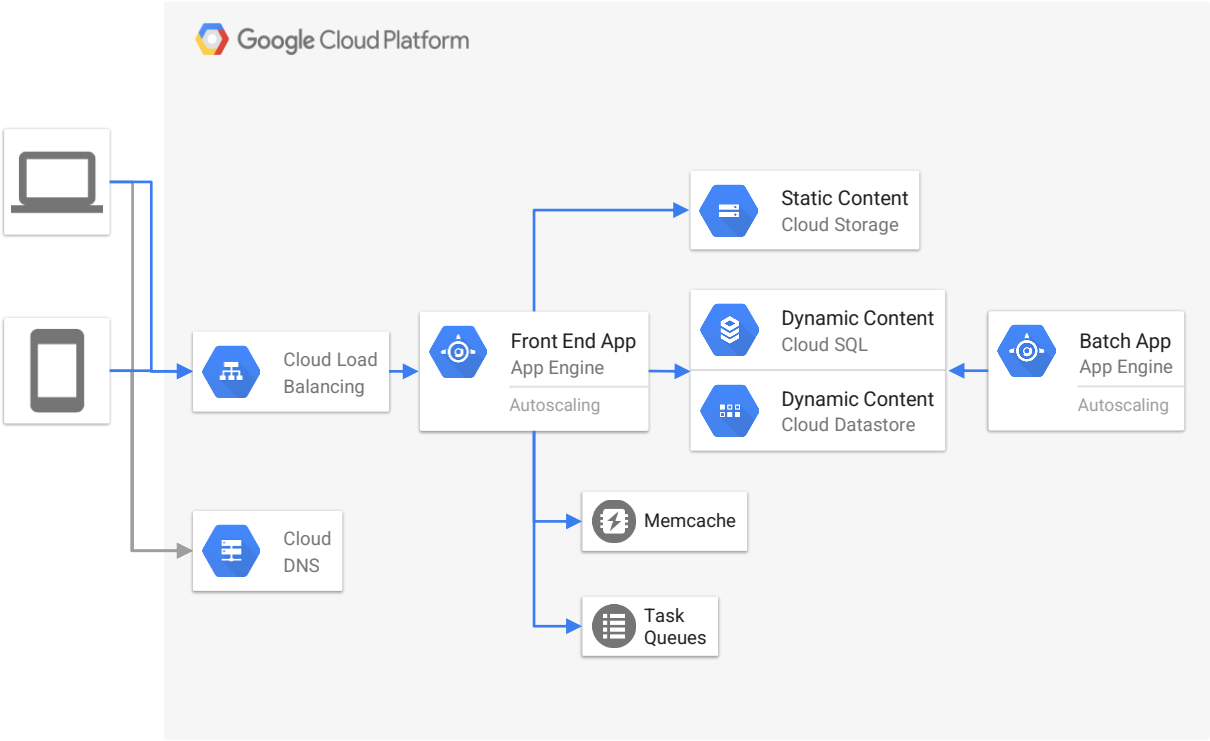
- Platform-as-a-Service (PaaS)
  - Fully managed serverless application platform
  - Server infrastructure and software installation is managed by Google
- Automatic scaling and load balancing
- Pay only for what you use
  - Free developer quota available
- Comparable to:

AWS Elastic Beanstalk Azure App Service Red Hat OpenShift

# Deploy your code in the cloud

- Choose one or more regions (39 available)
- One application is fixed to a region

# Example App Engine Deployment



# Programming languages

- Managed languages: zero configuration
  - Java, Go, PHP, Node.js, Python, Ruby
- Custom containers
  - .NET, Rust, Erlang, Haskell, Cobol, …

# Restrictions

- Sandbox environment
  - Limited access to underlying platform
  - Can only write to `/tmp`
  - Isolated from other applications
- App must not respond slowly
  - Web requests must be handled within 10 minutes

# Automatic scaling

- Instances are created on demand to handle requests
- Automatically turned down when idle
- Low latency ↔ costs
  - Cold-start problem: starting a new instance takes some time (seconds)
- In-memory state might not be present at the next request

# Writing Your 1st Java 17 Web Service (1/3)

- Directory structure:

```
pom.xml
src/
    main/
        appengine/
            app.yaml
        java/
            com/example/appengine/springboot/
                SpringbootApplication.java
```

https://cloud.google.com/appengine/docs/standard/java-gen2/building-app/writing-web-service



Level 1

# Writing Your 1st Java 17 Web Service (2/3)

src/main/java/com/example/appengine/springboot/SpringbootApplication.java

```java
@SpringBootApplication
@RestController
public class SpringbootApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringbootApplication.class, args);
  }

  @GetMapping("/")
  public String hello() {
    return "Hello world!";
  }
}
```

https://cloud.google.com/appengine/docs/standard/java-gen2/building-app/writing-web-service



Level 1

# Writing Your 1st Java 17 Web Service (3/3)

src/main/appengine/app.yaml

```
runtime: java17
```

- Local development:

```
$ mvn spring-boot:run
```

- Open your browser: `http://localhost:8080`

https://cloud.google.com/appengine/docs/standard/java-gen2/building-app/writing-web-service



Level 1

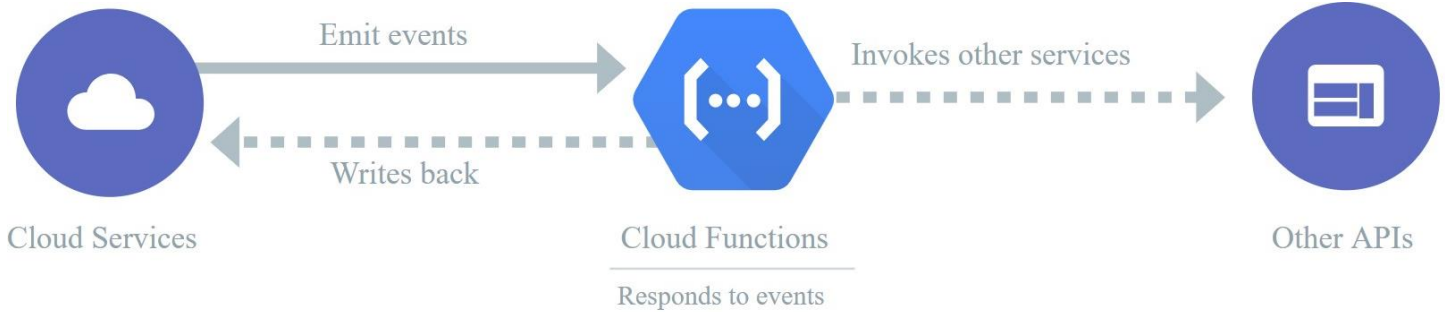
# What’s the difference with FaaS?

- Function-as-a-Service (FaaS)
  - A single function, not a full application
- Function can run everywhere, close to the user
- Automatic scaling, instantly
- Pay only what you use: fine-grained, i.e., 100 milliseconds
- Comparable to

AWS Lambda Azure Functions Cloudflare workers

# Cloud Functions

- One single piece of code, triggered by some event



# Can you host it in your own data center?

- Kubernetes
  - Open-source container orchestration
  - Built upon the experience of Google with App Engine
  - But only orchestration and deployments
    - No other built-in services

# Firebase Authentication

# What is Firebase Authentication?

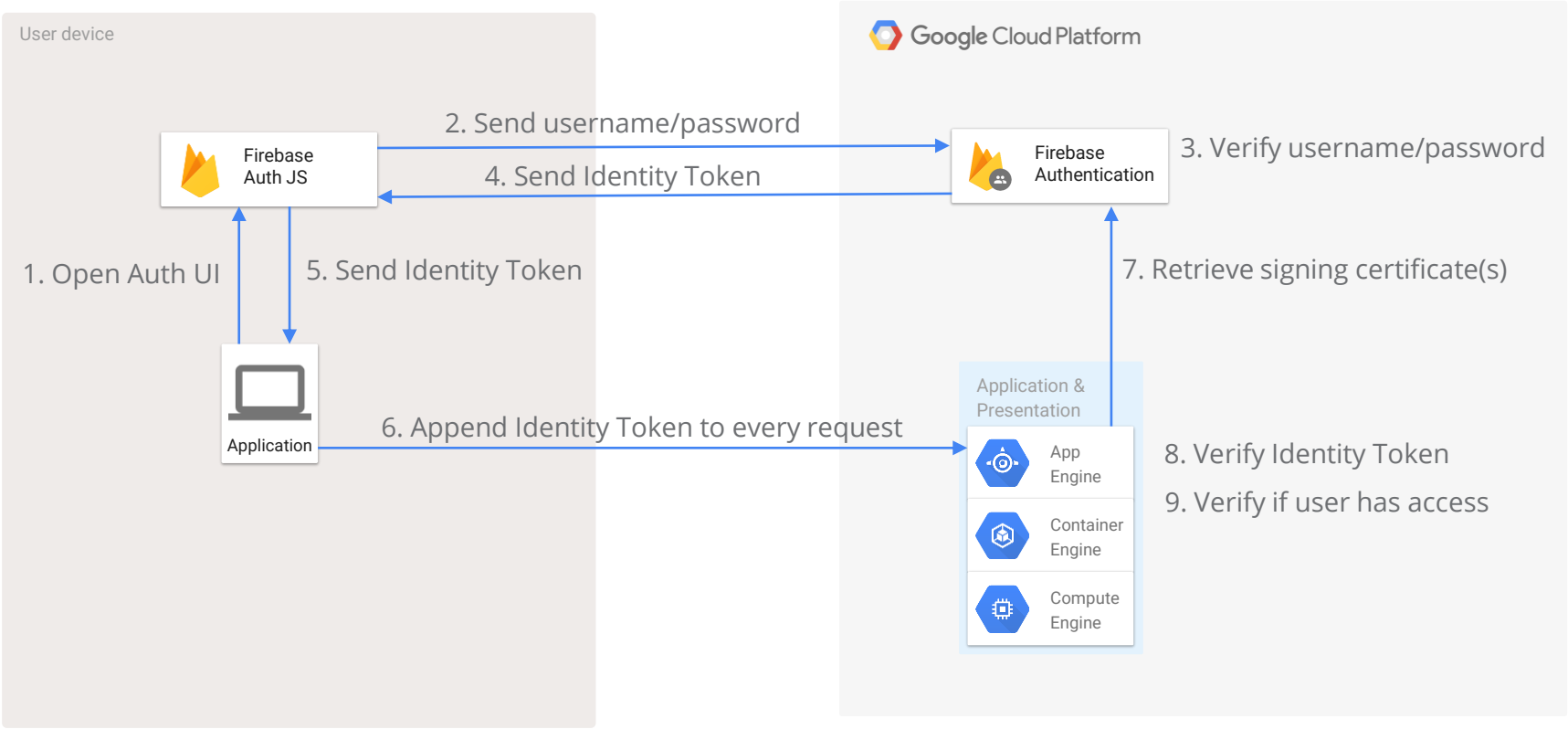
- Identity and Access Management Solution
  - Secure and scalable identity store
  - Social identity federation: Google, Facebook, Twitter, …
  - Standards-based authentication: OAuth 2.0, OpenID Connect
- Comparable to:

Amazon Cognito Azure AD Auth0

# OpenID Connect

- Authentication protocol
  - Securely login a user to an application
  - Built on OAuth 2.0
- Identity token
  - Proves identity of a user as verified by the identity provider

# Example Authentication flow



# OpenID Identity Token: raw format


```
eyJhbGciOiJSUzI1NiIsImtpZCI6IjhmYmRmMjQxZTdjM2E2NTEzNTYwNmRkYzFmZWQyYzU1MjI2MzBhODciLCJ0eXAiOiJKV1QifQ.eyJuYW1lIjoiS3Jpc3RvZiBKYW5uZXM
iLCJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vZGlzdHJpYnV0ZWQtc3lzdGVtcy1rdWwtMzI3MDA3IiwiYXVkIjoiZGlzdHJpYnV0ZWQtc3lzdGVtcy1rd
WwtMzI3MDA3IiwiYXV0aF90aW1lIjoxNjM1MDk4OTUzLCJ1c2VyX2lkIjoiZ2JSSWtSZlViOFlNMUQxZHROMWlWRnBuZ29jMiIsInN1YiI6ImdiUklrUmZVYjhZTTFEMWR0TjF
pVkZwbmdvYzIiLCJpYXQiOjE2MzUwOTg5NTQsImV4cCI6MTYzNTEwMjU1NCwiZW1haWwiOiJrcmlzdG9mLmphbm5lc0BrdWxldXZlbi5iZSIsImVtYWlsX3ZlcmlmaWVkIjpmY
WxzZSwiZmlyZWJhc2UiOnsiaWRlbnRpdGllcyI6eyJlbWFpbCI6WyJrcmlzdG9mLmphbm5lc0BrdWxldXZlbi5iZSJdfSwic2lnbl9pbl9wcm92aWRlciI6InBhc3N3b3JkIn1
9.gZLjVNMHIL-nDXAl4bmz0DEjCqIFDG91y5xqRadQnz8kfj02ZiuJvTSy0JJm41gXhoZP7462GVgaYR-_cxiscGydj4B_roxBniaspQ4vFqZFwTTZ2tJo29kS-pxn4Xo7dgHN
QYxQcWW15RGqcoUMkdHUDAPtMnRqP4u3lceL9ZsB0JZyXYNmNDQdidUmvxcSr2WnILzonSL98yG7OtuC6UQneH2opvNAy3WAy40li3s9UBLLhE-xtJj3OvNWVB7KJZdICmbHWX
SBipS5yBCgSG-Jn9CpXwiHYn584u5_4xIPj-18td28oNjqBhO6ryecrfWxtXZ6OVPWUY6sQsFicw
```


https://jwt.io/ https://jwt.ms/

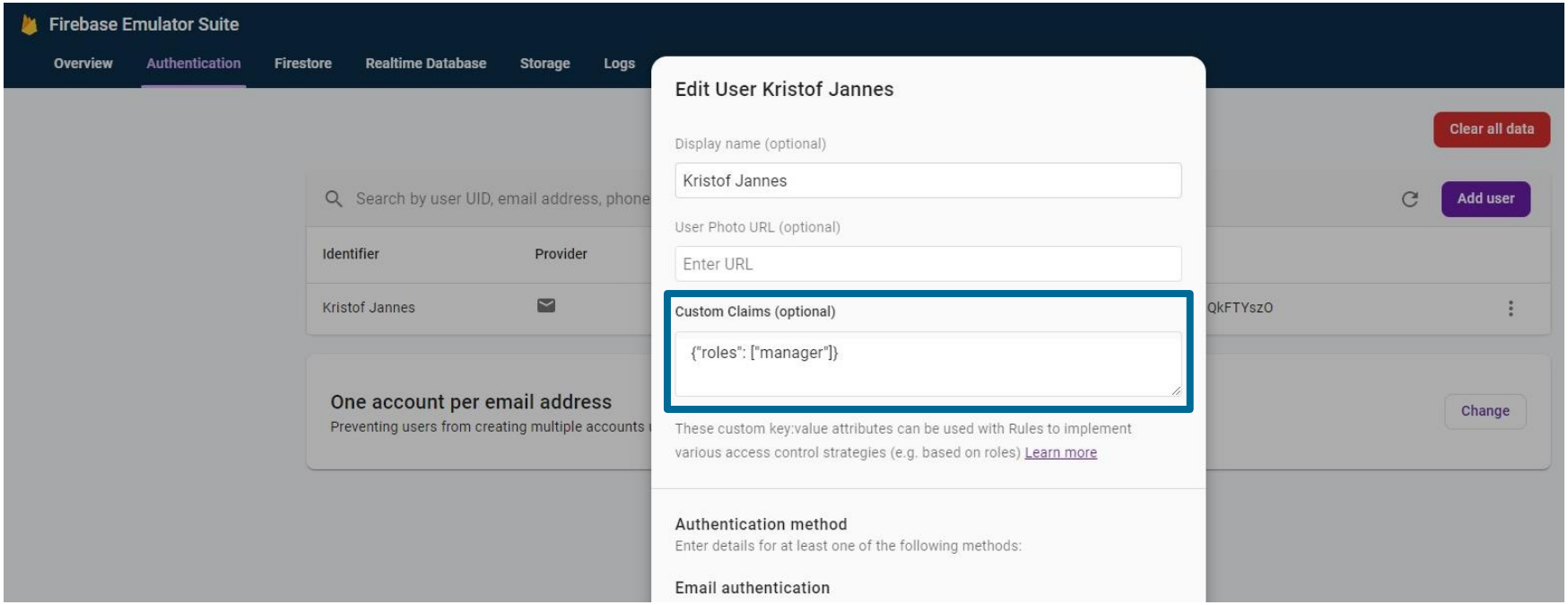
# OpenID Identity Token: decoded

3 parts:

- **Header**
  - contains details about how the JWT token is signed
- **Payload**
  - contains claims
- **Signature**


```
{
  "alg": "RS256",
  "kid": "8fbdf241e7c3a65135606ddc1fed2c5522630a87",
  "typ": "JWT"
}.{
  "name": "Kristof Jannes",
  "iss": "https://securetoken.google.com/distributed-systems-kul-327007",
  "aud": "distributed-systems-kul-327007",
  "auth_time": 1635098953,
  "user_id": "gbRIkRfUb8YM1D1dtN1iVFpngoc2",
  "sub": "gbRIkRfUb8YM1D1dtN1iVFpngoc2",
  "iat": 1635098954,
  "exp": 1635102554,
  "email": "kristof.jannes@kuleuven.be",
  "email_verified": false,
  "firebase": {
    "identities": {
      "email": [
        "kristof.jannes@kuleuven.be"
      ]
    },
    "sign_in_provider": "password"
  }
}.[Signature]
```

# Add custom claims to users






Level 1

# Custom claims are added to Identity Token


```
{
  "alg": "RS256",
  "kid": "8fbdf241e7c3a65135606ddc1fed2c5522630a87",
  "typ": "JWT"
}.{
  "name": "Kristof Jannes",
  "iss": "https://securetoken.google.com/distributed-systems-kul-327007",
  "aud": "distributed-systems-kul-327007",
  "auth_time": 1635098953,
  "user_id": "gbRIkRfUb8YM1D1dtN1iVFpngoc2",
  "sub": "gbRIkRfUb8YM1D1dtN1iVFpngoc2",
  "iat": 1635098954,
  "exp": 1635102554,
  "email": "kristof.jannes@kuleuven.be",
  "email_verified": false,
  "firebase": {
    "identities": {
      "email": [
        "kristof.jannes@kuleuven.be"
      ]
    },
    "sign_in_provider": "password"
  },
  "roles": ["manager"]
}.[Signature]
```

# Verify Identity Token

```java
try {
  var kid = JWT.decode(idToken).getKeyId();
  var pubKey = PUBLIC_KEYS.get(kid);
  Algorithm algorithm = Algorithm.RSA256(pubKey, null);
  DecodedJWT jwt = JWT.require(algorithm)
                          .withIssuer("https://securetoken.google.com/" + projectId)
                          .build()
                          .verify(idToken);
  var email = jwt.getClaim("email");
} catch (JWTVerificationException e) {
  // unauthorized
}
```

https://github.com/auth0/java-jwt

Level 2

# How difficult is getting API security right?

https://owasp.org/www-project-api-security/

T10 **OWASP API Security Top 10 - 2019**

| | |
|---|---|
| API1:2019 - Broken Object Level Authorization | APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user. |
| API2:2019 - Broken User Authentication | Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising system's ability to identify the client/user, compromises API security overall. |
| API3:2019 - Excessive Data Exposure | Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user. |
| API4:2019 - Lack of Resources & Rate Limiting | Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force. |
| API5:2019 - Broken Function Level Authorization | Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions. |
| API6:2019 - Mass Assignment | Binding client provided data (e.g., JSON) to data models, without proper properties filtering based on a whitelist, usually lead to |

OWASP®

# Cloud Pub/Sub

# What is Cloud Pub/Sub?

- Indirect communication
- Asynchronous communication, outside user request
- Decouple services (micro-service architecture)
- Scale and parallelize execution: flow control
- Comparable to:

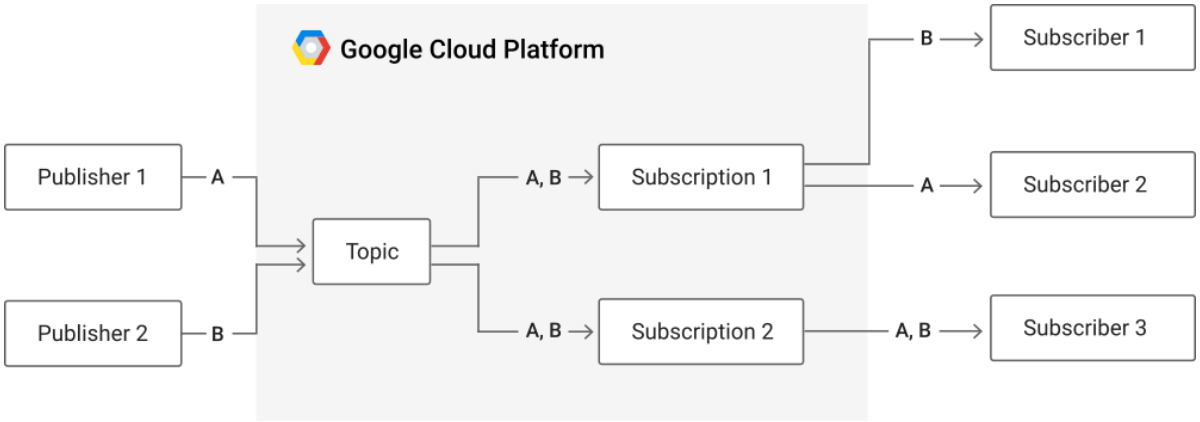
Amazon SNS  Azure Queues  RabbitMQ

# Core concepts

- **Topic**: unique name
- **Publisher**: sends messages
- **Subscriber**: receives messages

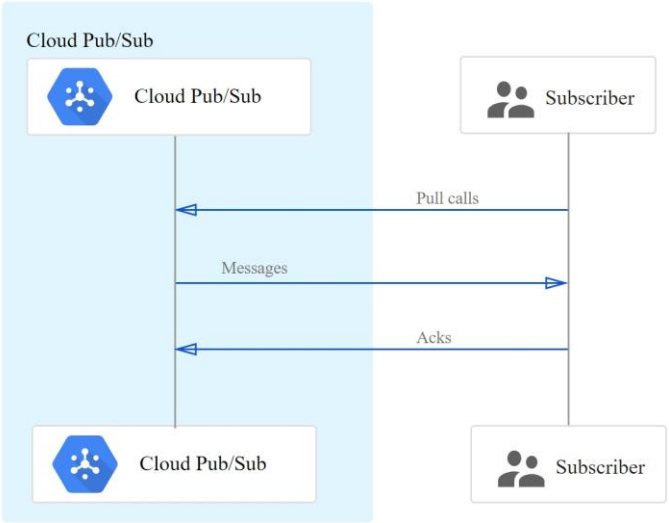
2 delivery methods:

- Pull
- Push



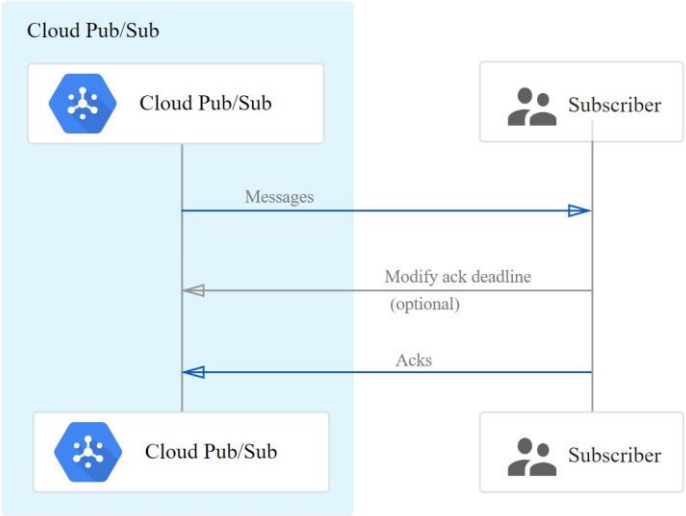
# Pull subscription

Application asks Cloud Pub/Sub for next message



# Push subscription

Cloud Pub/Sub sends message to pre-defined endpoint



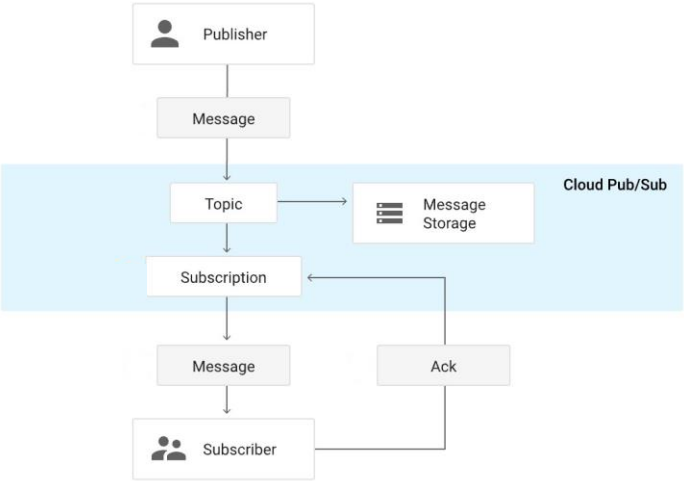
# Delivery semantics

- **At-least-once** delivery
  - Duplicate messages are possible!
- Subscriber should acknowledge successful processing
  - Otherwise, message is retried (up to 7 days)

# How does Pub/Sub guarantee at-least-once delivery?

Lifecycle of a message:

1. A publisher sends a message.
2. The message is written to **durable storage**.
3. Pub/Sub sends an acknowledgement to the publisher that it has received the message and guarantees its delivery to all attached subscriptions.
4. At the same time as writing the message to storage, Pub/Sub delivers it to subscribers.
5. Subscribers send an acknowledgement to Pub/Sub that they have processed the message.
6. Once at least one subscriber for each subscription has **acknowledged** the message, Pub/Sub deletes the message from storage.



# Publishing a message

```
TopicName topicName = TopicName.of(projectId, topicId);
Publisher publisher = Publisher.newBuilder(topicName).build();

PubsubMessage pubsubMessage = PubsubMessage.newBuilder()
                                            .setData(data)
                                            .putAttributes(key, value)
                                            .build();
ApiFuture<String> future = publisher.publish(pubsubMessage);
```

https://cloud.google.com/pubsub/docs/quickstart-client-libraries#publish_messages



Level 1

# Create push subscription

```
SubscriptionAdminClient subscriptionAdminClient = SubscriptionAdminClient.create();
PushConfig pushConfig = PushConfig.newBuilder()
                            .setPushEndpoint("http://url.of.subscriber.endpoint")
                            .build();
subscriptionAdminClient.createSubscription(subscriptionName, topicName, pushConfig, 60);
```

Acknowledgement deadline

https://cloud.google.com/pubsub/docs/create-push-subscription#create_a_push_subscription



Level 1

# Receive message from push subscription

```
@RestController
public class Controller {

  @PostMapping("/subscription")
  public void subscription(@RequestBody String body) {
    // answer with 2xx HTTP status code to acknowledge
  }
}

```

https://cloud.google.com/pubsub/docs/push#receive_push

# Anatomy of a message

Base64 representation of the data you published

```
{
  "message": {
    "attributes": {
      "key": "value"
    },
    "data": "SGVsbG8gQ2xvdWQgUHViL1N1YiEgSGVyZSBpcyBteSBtZXNzYWdlIQ==",
    "messageId": "2070443601311540",
    "message_id": "2070443601311540",
    "publishTime": "2021-02-26T19:13:55.749Z",
    "publish_time": "2021-02-26T19:13:55.749Z"
  },
  "subscription": "projects/myproject/subscriptions/mysubscription"
}
```

https://cloud.google.com/pubsub/docs/push#receive_push



Level 1

# Cloud Firestore

# What is Cloud Firestore?

- Fast, fully managed, serverless, cloud-native NoSQL document database
- Highly scalable
- Highly available
- Comparable to:

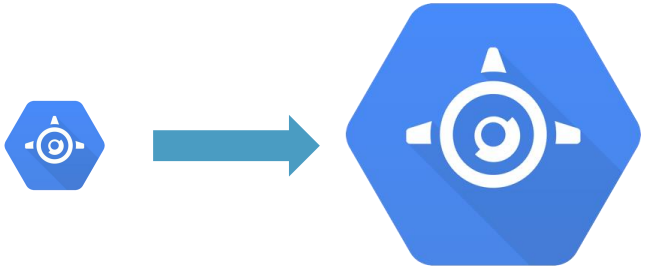
MongoDB AWS DynamoDB Azure Cosmos DB

# NoSQL document database

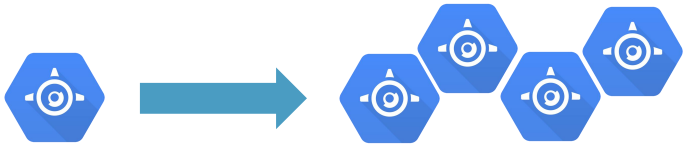
- ⇔ relational database management systems (RDBMS)
- Schemaless: no fixed fields
- Fast and scalable
  - Often weak consistency guarantees (not Firestore)
    - Eventual consistency
    - Only transactions on single data items
  - Often limited query possibilities
- Horizontal scalability

# Horizontal scalability

- Vertical
  - Add resources to single node



- Horizontal
  - Add more nodes



# Data model

- Documents ~row
  - Key-value pairs / very similar to JSON
  - Unique reference
  - No schema
- Collections ~table
  - Collection of documents
  - Hierarchical model ~one-to-many

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

alovelace

```
name :
  first : "Ada"
  last : "Lovelace"
born : 1815
```

users

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

aturing

```
first : "Alan"
last : "Turing"
born : 1912
```

rooms

roomA

```
name : "my chat room"
```

messages

message1

```
from : "alex"
msg : "Hello World!"
```

message2

```
...
```

roomB

```
...
```

# References

- Identifies one unique document

users

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

aturing

```
first : "Alan"
last : "Turing"
born : 1912
```

```
DocumentReference document = db.collection("users").document("alovelace");
```



Level 1

# Modifying documents

- Add / Edit:

```
Map<String, Object> docData = new HashMap<>();
docData.put("first", "Ada");
docData.put("last", "Lovelace");
docData.put("born", 1815);
ApiFuture<WriteResult> future = db.collection("users").document("alovelace").set(docData);
future.get(); // block until operation is completed
```

- Delete:

```
db.collection("users").document("alovelace").delete().get();
```

users

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

aturing

```
first : "Alan"
last : "Turing"
born : 1912
```

# Retrieving documents

- Single document:

```
db.collection("users").document("alovelace").get().get()
```

- All documents:

```
db.collection("users").get().get().getDocuments()
```

users

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

aturing

```
first : "Alan"
last : "Turing"
born : 1912
```

# Queries

- Limited query support
  - No many-to-many relations
  - No JOIN queries
  - No aggregate queries
  - Inequality filters are limited to at most one field
- Think about performance
  - If you know the ID of a document, a simple `get` is much faster

# Queries

- Examples:

```
Query query = db.collection("users").whereEqualTo("last", "Lovelace");
ApiFuture<QuerySnapshot> querySnapshot = query.get();
for (DocumentSnapshot document : querySnapshot.get().getDocuments()) {
    ...
}
```

```
Query query = db.collection("users").whereLessThan("born", 1900);
ApiFuture<QuerySnapshot> querySnapshot = query.get();
for (DocumentSnapshot document : querySnapshot.get().getDocuments()) {
    ...
}
```

users

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

aturing

```
first : "Alan"
last : "Turing"
born : 1912
```



Level 2

# Transactions

- Pessimistic concurrency control: locks
- Serializable isolation
  - You can assume that the database executes transactions in series
  - Transactions are not affected by uncommitted changes in concurrent operations

# Transactions

```
final DocumentReference docRef = db.collection("users").document("alovelace");
db.runTransaction(transaction → {
  DocumentSnapshot snapshot = transaction.get(docRef).get();
  long oldBalance = snapshot.getLong("balance");
  transaction.update(docRef, "balance", oldBalance + 1);
  return null;
});
```



Level 2

# Transactions

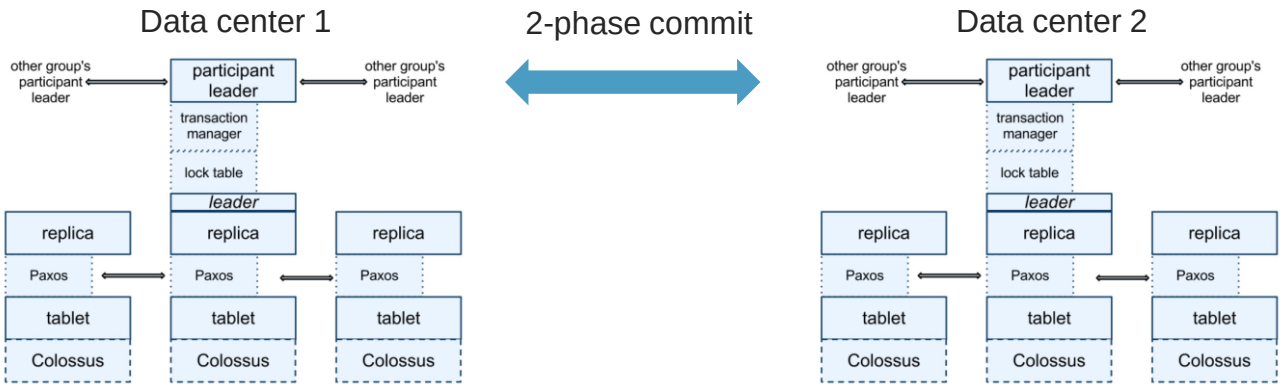
- Limitations:
  - Read operations must come before write operations
  - A transaction function might run more than once

```
db.runTransaction(transaction → ... )
```

  - At most 500 documents
- How to solve resource contention?
  - Indirect communication ➔ Cloud Pub/Sub (flow control)

# How are transactions executed under the hood?

- Documents are distributed across servers and possibly data-centers
  - Based on collection and key
- 2-phase commit is needed for transactions across multiple data centers

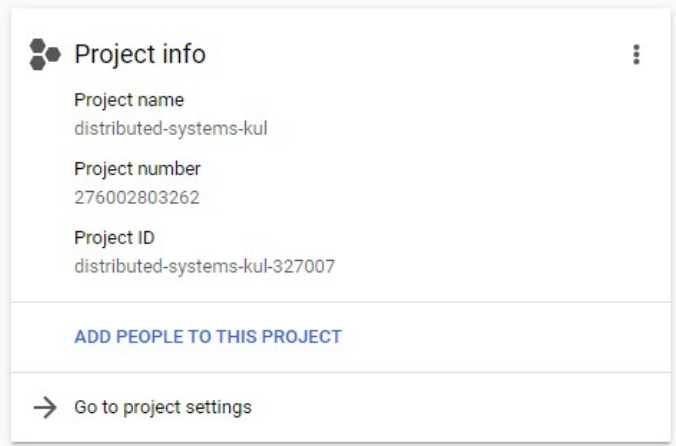



[Firestore: The NoSQL Serverless Database for the Application Developer]
[Spanner: Google's Globally-Distributed Database]

# Google Cloud Platform Deployment

# Google Cloud Project

- Each Cloud resource must belong to a project
- Each project ID is unique across Google Cloud Platform



# Select geographic zones and regions

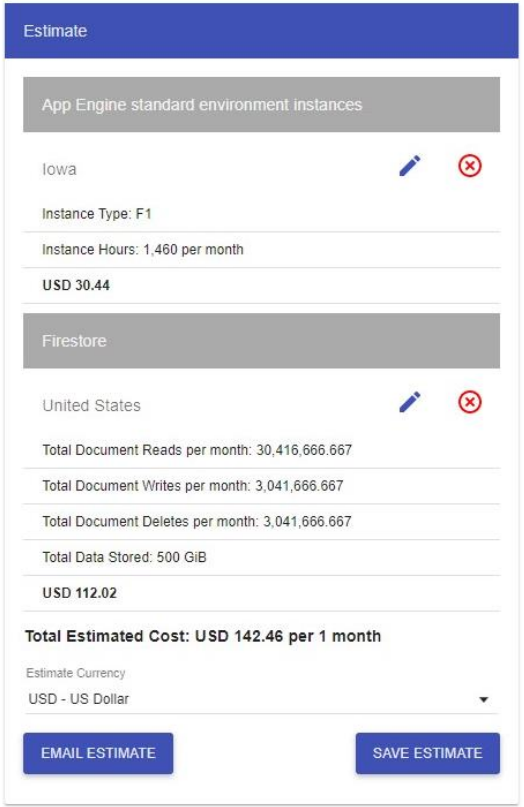
- Based on:
  - Geographic proximity
  - Available services
  - Compliance
  - Price
- Some services: multi-region
  - More expensive, but more reliable

# Serving websites

- Each Google App Engine receives a unique URL
  - https://<project-id>.<region>.r.appspot.com
- SSL certificate managed by Google
- Possible to buy custom domain

# Billing

- To use Google Cloud services,
  you must have a valid Cloud Billing account
  - Project is linked to one billing account
  - *We provide credits for the optional lab sessions*
- Some services: free quota
- Planning: Google Cloud Pricing Calculator
  - https://cloud.google.com/products/calculator



# Deploy to Google App Engine

- Package all files into a WAR file

```
$ mvn package
```

- Upload and deploy to Google Cloud

```
$ mvn appengine:deploy
```

- Other services must be configured through the cloud console
  - https://console.cloud.google.com



Level 2

# Spring Boot

# Component-based development in Java

- Objects and their relations are managed by the framework
- Framework automatically loads correct objects and services
- Dependency injection
  - Automatically inject correct object based on rules
  - Decouples part of the application
- Example frameworks:

# Booting the application

- Component scanning
  - Framework will discover all Spring Components and start them

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

# Declaring components (1/2)

- `@Component`
  - Registers a class as a component
  - An instance will automatically be created when the application starts

```
@Component
public class SecurityFilter extends OncePerRequestFilter {
    ...
}
```

# Declaring components (2/2)

- `@Bean`
  - Registers the result of a method as a component (also called *bean*)
  - Object is managed by Spring, and can be injected in other components

```java
@SpringBootApplication
public class Application {

    @Bean
    public Firestore db() {
        return FirestoreOptions.getDefaultInstance()
                .toBuilder()
                .setProjectId(this.projectId())
                .build()
                .getService();
    }

}
```



Level 1

# Building a REST API

- `@RestController`
  - Is a special component
  - Remember the REST lab sessions

```
@RestController
@RequestMapping("/api")
public class APIController {

    @GetMapping("/getTrains")
    public List<Train> getTrains() {
        ...
    }

}
```



Level 1

# Dependency Injection (1/2)

- Inject a component or bean by its name
  - Spring will look up the bean, and instantiate it when necessary

```java
@SpringBootApplication
public class Application {

    @Bean
    public Firestore db() {
        return FirestoreOptions.getDefaultInstance()
                    .toBuilder()
                    .setProjectId(this.projectId())
                    .build()
                    .getService();
    }

}

```

```java
@RestController
@RequestMapping("/api")
public class APIController {

    @Resource(name = "db")
    private Firestore db;

}
```



Level 1

# Dependency Injection (2/2)

- Inject a component by its type
  - Spring will look up the component, and instantiate it when necessary

```java
@Component
public class Model {

  @Resource(name = "db")
  private Firestore db;
}
```

```java
@RestController
public class ViewController {

  private final Model model;

  @Autowired
  public ViewController(Model model) {
      this.model = model;
  }
}
```



Level 1

# Dependency injection happens after instantiation

- Constructor is called before any dependencies are injected
- Use `@PostConstruct` for initialization using dependencies

```
@Component
public class APIController {

    @Resource(name = "db")
    private Firestore db;

    constructor() {
        this.db.do_something(); // NullPointerException
    }

    @PostConstruct
    public init() {
        this.db.do_something(); // ok
    }
}
```



Level 1

# Spring WebClient

```
curl -X GET https://reliabletrains.com/trains?key= ...
```

=

```
WebClient.Builder webClientBuilder; // dependency injection
var trains = this.webClientBuilder
                    .baseUrl("https://reliabletrains.com")
                    .build()
                    .get()
                    .uri(uriBuilder → uriBuilder
                            .pathSegment("trains")
                            .queryParam("key", API_KEY)
                            .build())
                    .retrieve()
                    .bodyToMono(new ParameterizedTypeReference<CollectionModel<Train>>() {})
                    .block()
                    .getContent();
```

https://docs.spring.io/spring-framework/reference/web/webflux-webclient/

# Lab Sessions

# Recap lab sessions

- Remote communication: completed
  - **Deliverables: report (20 Oct)**
- Distributed cloud applications
  - Mandatory (level 1): 24 Oct, 31 Oct, 7 Nov, 14 Nov
    - Topics: PaaS, Access Control, Indirect Communication, Fault Tolerance, NoSQL
    - Deliverables: code + report (**17 Nov**)
  - Optional (level 2): 21 Nov, 28 Nov, 5 Dec
    - Topics: NoSQL, Transactions, Cloud Deployment, Scalability
    - Deliverables: code + report (8 Dec)

# Assignment

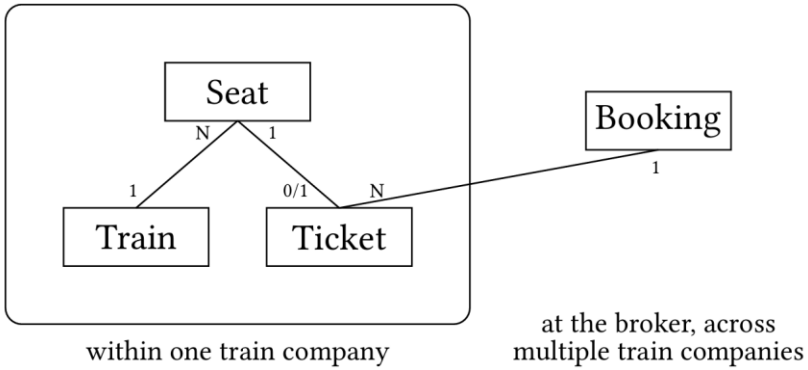
- Build the backend of a **booking platform** for making reservations across **multiple train companies**

- Comparable to:

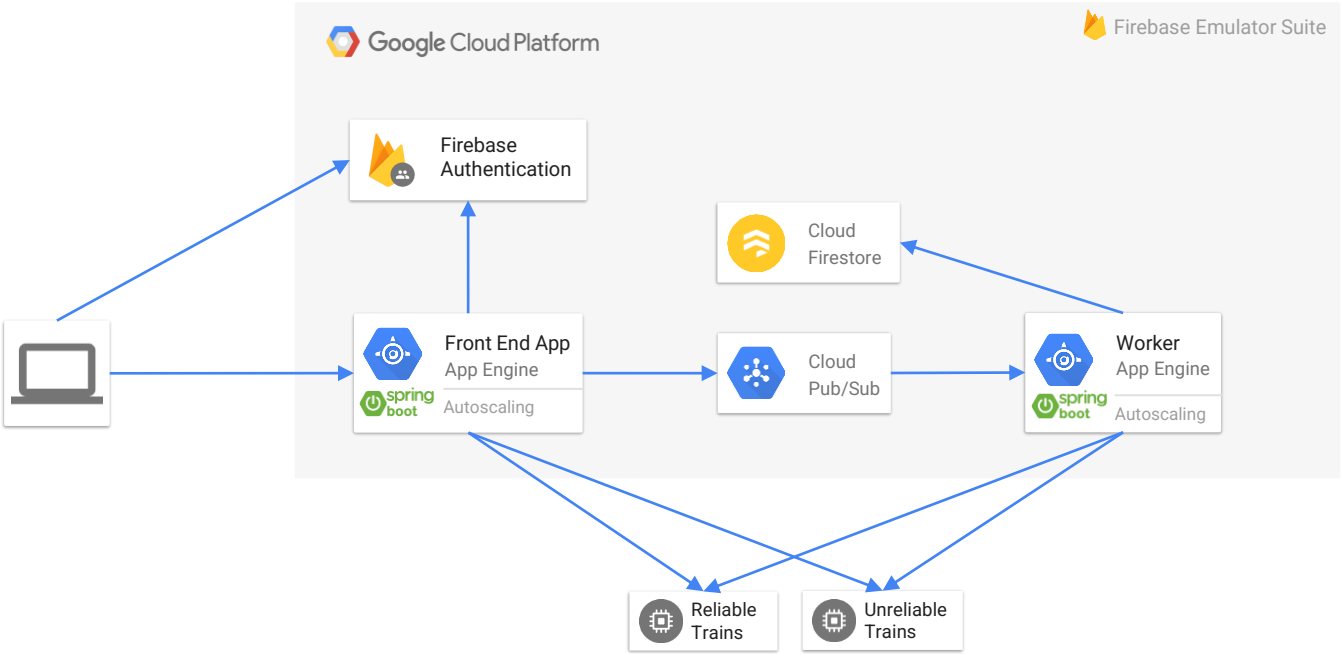
Welcome to
SNCB International

# Domain model

- Train Company
  - **Train**: a train with name and destination
  - **Seat**: a specific instance of a train with time and seat number
  - **Ticket**: a reservation for a specific seat
- Booking Platform
  - **Booking**: a collection of tickets

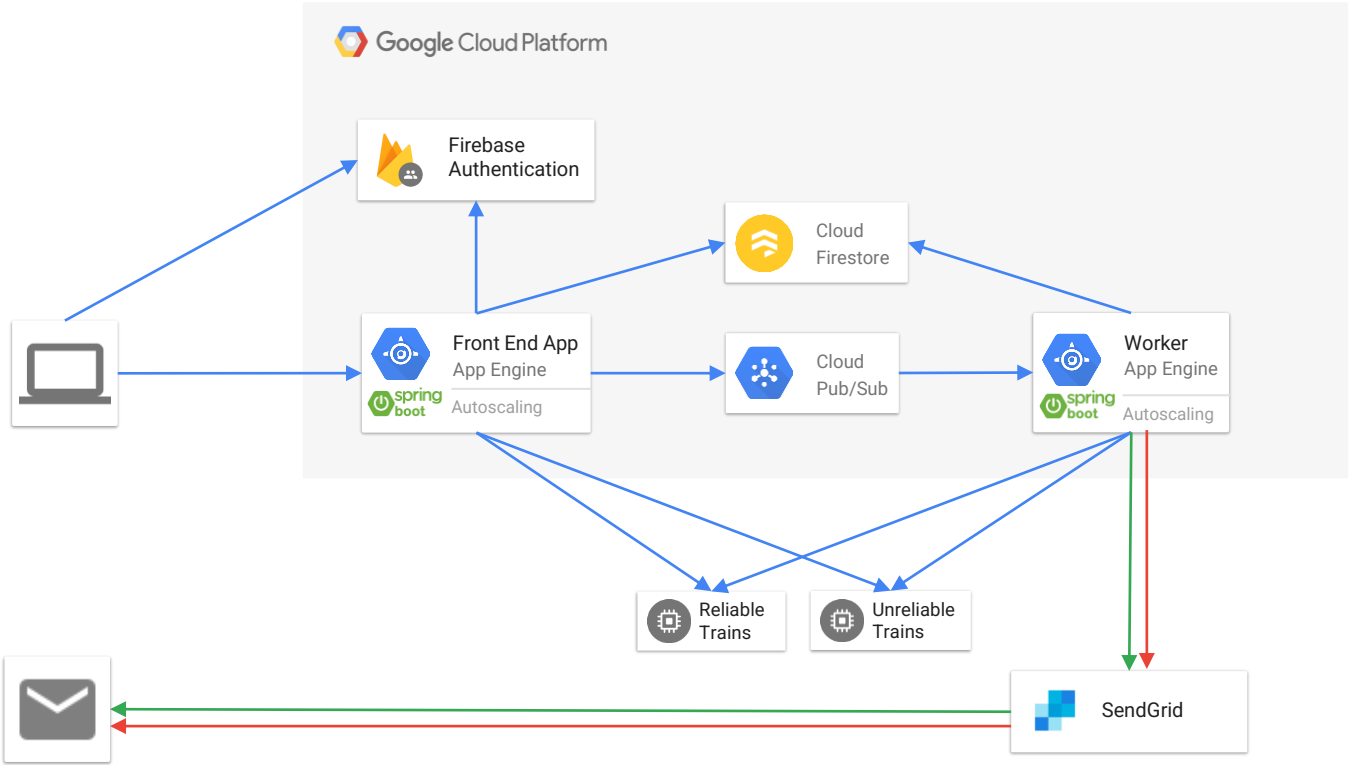


# Goal: level 1 (mandatory)




Level 1

# Goal: level 2 (optional)




Level 2

# Assignment: level 1 (mandatory)

- Provided:
  - Spring Boot application with Preact
  - 2 external train companies with RESTful API
- Requirements:
  - Implement business logic by using the REST endpoints of the train companies
  - Implement authentication and authorization through Firebase Authentication
  - Use indirect communication through Cloud Pub/Sub to decouple reservation processing
  - Maintain ACID semantics
  - Ensure that the platform is fault tolerant even when relying on an unreliable service
  - Test your application locally

# Provided web-app will use REST to call your service

```yaml
openapi: 3.0.3
info:
  title: DNetTickets - API
  version: 3.0.0
paths:
  /api/getTrains:
    get:
      summary: Get all trains
      responses:
        "200":
          description: A list of trains
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: "#/components/schemas/Train"
              example:
                - trainCompany: reliable-trains.com
                  trainId: de9bdc62-d08c-4ac2-8838-cfba02e556f0
                  name: Thalys Amsterdam
                  location: Brussels - Amsterdam
                  image: https://reliable-trains.com/amsterdam.jpg
  /api/getTrain:
    get:
      summary: Get train by ID
      parameters:
        - name: trainCompany
          in: query
          description: ID of the train company
          required: true
          schema:
```

...

| | | |
|---|---|---|
| GET | /api/getTrains | Get all trains |
| GET | /api/getTrain | Get train by ID |
| GET | /api/getTrainTimes | Get all times for a train |
| GET | /api/getAvailableSeats | Get all available seats for a train at a specific time |
| GET | /api/getSeat | Get seat by ID |
| POST | /api/confirmQuotes | Create a new booking |
| GET | /api/getBookings | Get bookings from customer |
| GET | /api/getAllBookings | Get all bookings |
| GET | /api/getBestCustomers | Get the best customer |

OPENAPI INITIATIVE

https://editor.swagger.io/

Level 1

# Local development


```
$ firebase emulators:start --project demo-distributed-systems-kul
```

# Remember to connect to the emulator

```java
@SpringBootApplication
public class Application {

    @Bean
    public Publisher publisher() throws IOException {
        TransportChannelProvider channelProvider = FixedTransportChannelProvider.create(
                    GrpcTransportChannel.create(
                            ManagedChannelBuilder.forTarget("localhost:8083").usePlaintext().build()));
        CredentialsProvider credentialsProvider = NoCredentialsProvider.create();
        return Publisher
                    .newBuilder( ... )
                    .setChannelProvider(channelProvider)
                    .setCredentialsProvider(credentialsProvider)
                    .build();
    }

}
```

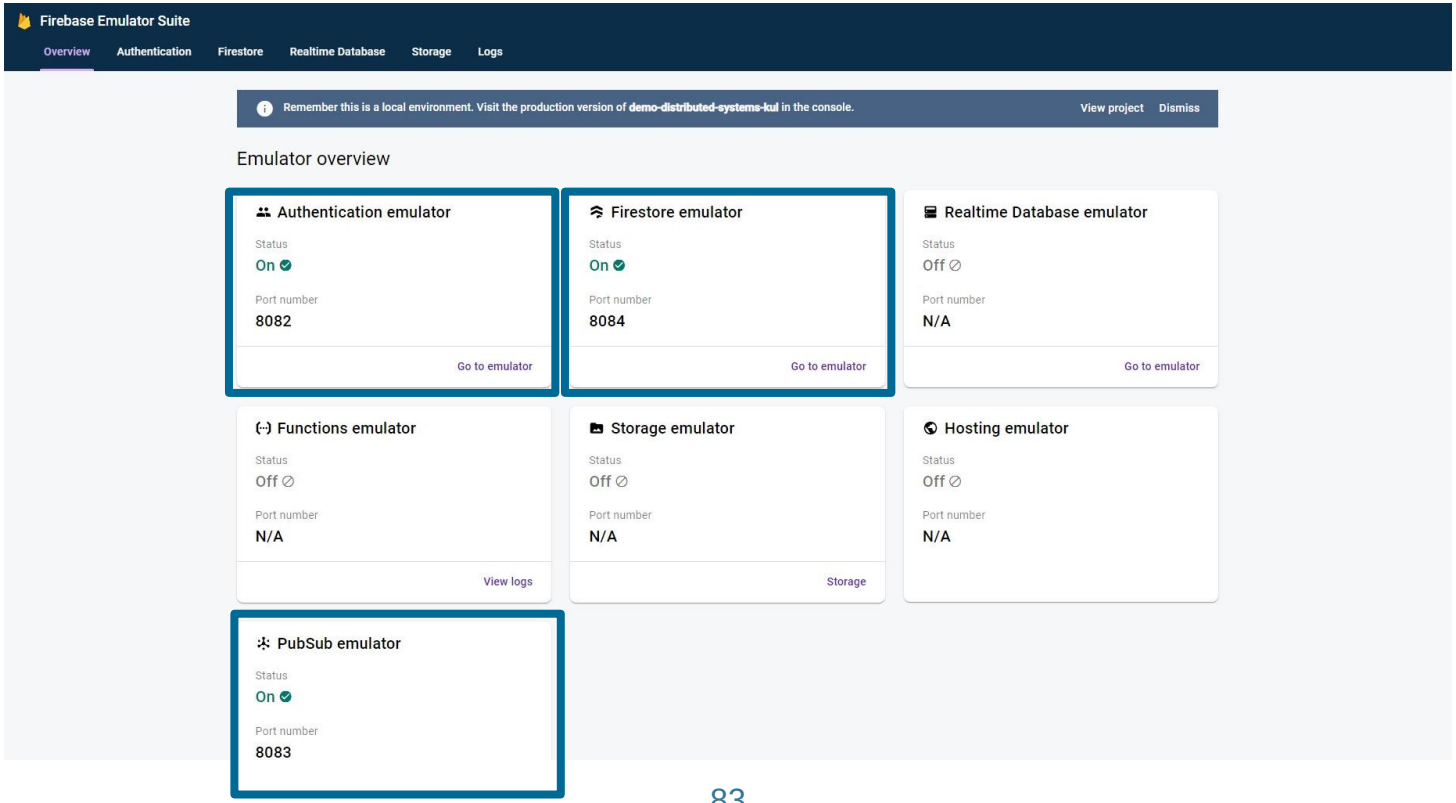
# Remember to connect to the emulator

```java
@SpringBootApplication
public class Application {

    @Bean
    public Firestore db() {
        return FirestoreOptions.getDefaultInstance()
                .toBuilder()
                .setProjectId(this.projectId())
                .setCredentials(new FirestoreOptions.EmulatorCredentials())
                .setEmulatorHost("localhost:8084")
                .build()
                .getService();
    }
}
```

# Local development

http://localhost:8081



# Demo

# How to prepare for Cloud – level 1 (mandatory)?

- To read:
  - Lecture *Cloud: Google Cloud*: 17/10 – slides on Toledo
  - Assignment text: on Toledo
- To look through:
  - Live demo: 17/10
  - Assignment text contains lots of links to documentation – use them!
  - Google Cloud Pub/Sub Tutorials:
    https://cloud.google.com/pubsub/docs/tutorials
- Theory: Lecture *Indirect Communication*

# Questions?

- Via *distributedsystems@cs.kuleuven.be*

- Not an online help desk!
  - Content questions → only in the **lab sessions**
    or on the discussion board on Toledo
  - Administrative issues → via email

DistriNet