Computação na nuvem
ISEL – LEIRT / LEIC / LEIM

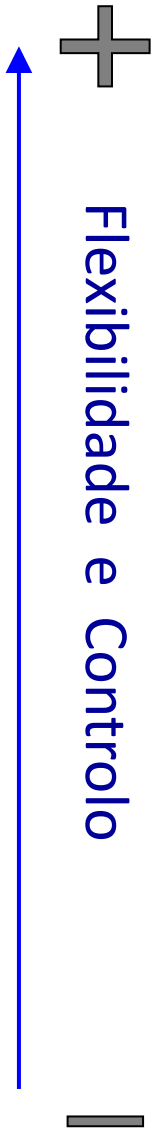
Serverless Computing (FaaS)

José Simão jsimao@cc.isel.ipl.pt ; jose.simao@isel.pt

Luís Assunção lass@isel.ipl.pt ; luis.assuncao@isel.pt

Revisão dos ambientes de execução

- Máquinas virtuais
  - Instâncias regulares ou preemptivas com diferentes recursos
  - Executam qualquer aplicação ou conjunto de aplicações
- *Containers*
  - Execução de imagens de software pré-construídas
  - Executam qualquer aplicação ou conjunto de aplicações
- Web Containers
  - Ambientes de execução para aplicações web
  - Executam várias operações (ex: rotas de uma web app)
- Funções
  - Funções desencadeada por chamadas a endpoint HTTP ou eventos de outros serviços, por exemplo *Storage*, Pub/Sub etc.
  - Executam uma operação por cada evento



Containers

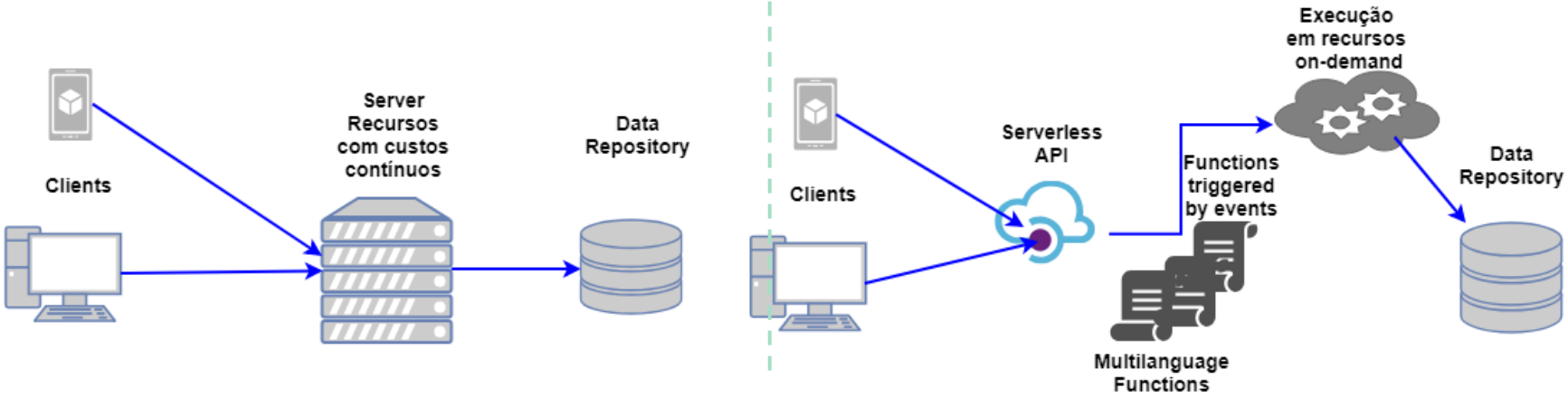
- Os *containers* são, no essencial, processos que correm no contexto do sistema operativo, geridos por um *runtime*
- Fornecem um isolamento inferior ao das VMs mas superior ao de processos regulares do sistema operativo, virtualizando o acesso ao sistema de ficheiros e utilizando os recursos (CPU, Mem, I/O) de um sistema operativo *host*, com algumas restrições
- Os *containers* executam imagens binárias, previamente construídas, e comprometidas com uma *Application Binary Interface* (ABI) (ex: linux, *windows*)
- Existem várias concretizações de *runtimes* de *containers*:
  - LXC, LXD, RKT, CRI-O, Docker …

Event-driven serverless computing

- Execução de código em múltiplas linguagens unicamente quando é necessário e com escalabilidade automática.
- O custo incorre unicamente durante o período de execução do código.

Computação tradicional: O servidor está ativo e a consumir recursos mesmo na ausência de eventos

Computação *serverless*: O servidor só está ativo e a consumir recursos durante o processamento do evento.



- No modelo *serverless* é importante que o código a executar seja *stateless* porque o número de instâncias disponíveis e o seu tempo de vida é gerido pela infraestrutura

Exemplos de serviços Cloud e *open source*

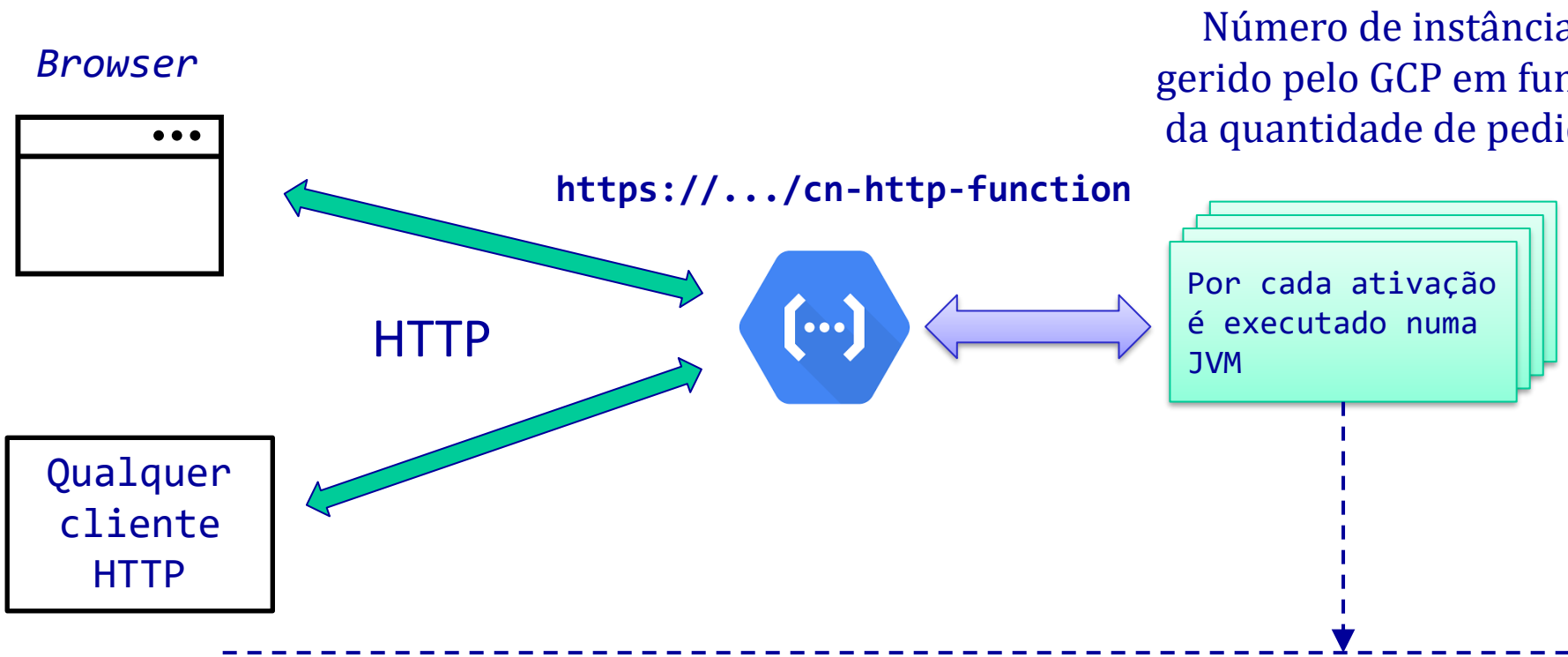
- Azure Functions
  - https://docs.microsoft.com/en-us/azure/azure-functions/
  - Código em C#, Java, JavaScript, Python, and PowerShell

- AWS Lambda
  - https://aws.amazon.com/lambda/
  - Código em Java, Go, PowerShell, Node.js, C#, Python e Ruby

- Google Cloud Functions
  - https://cloud.google.com/functions
  - Código em NodeJS, Python, Go e Java

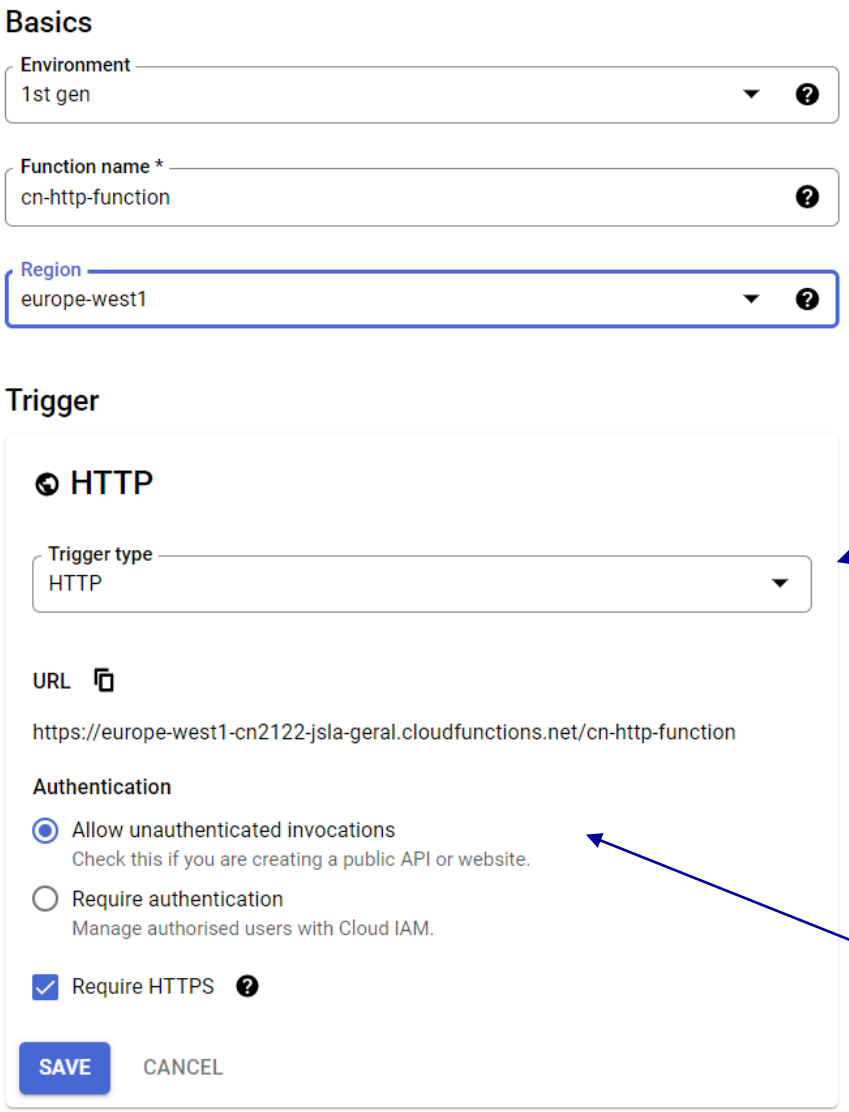
- Open Apache OpenWhisk
  - https://openwhisk.apache.org/
  - NodeJS, Go, Java, Scala, PHP, Python, Ruby, Swift, Ballerina, .NET e Rust.
  - Produto comercial IBM Cloud Functions baseado no Apache OpenWhisk

Invocação com *Trigger* HTTP

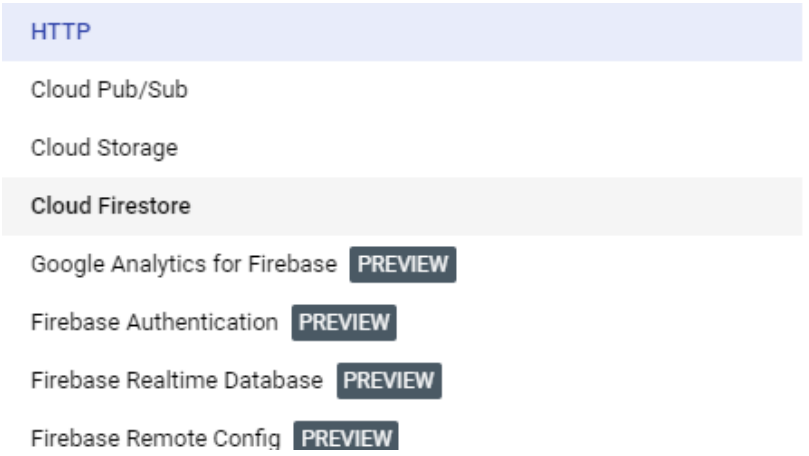



```
public class Example implements HttpFunction {
  @Override
  public void service(HttpRequest request, HttpResponse response) throws Exception
  {
    BufferedWriter writer = response.getWriter();
    writer.write("Hello world!");
  }
}
```

Google Cloud Functions

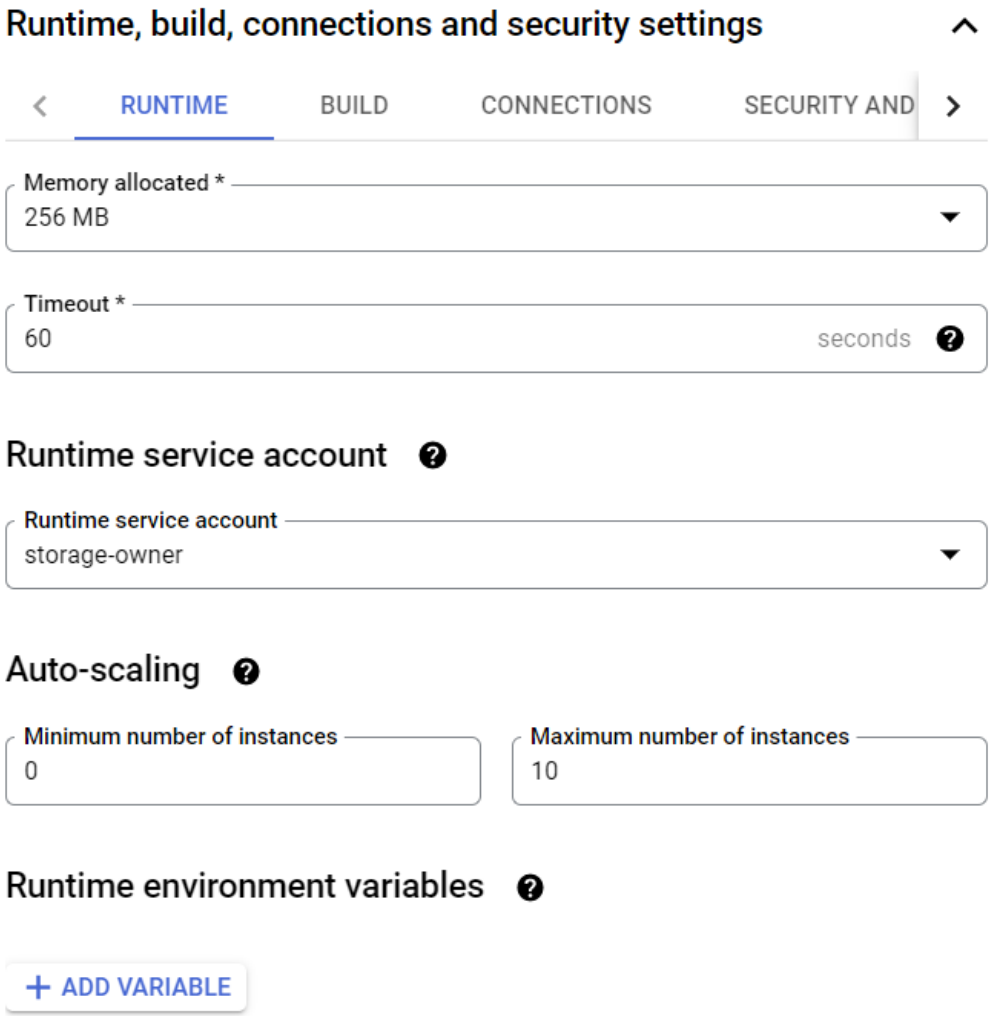


- Nome da *Cloud Function* (ex: `cn-http-function`) e região onde as instâncias serão alocadas
- *Trigger* do tipo HTTP: Um pedido HTTP desencadeia a execução da *Cloud Function*
  - Há vários tipos de *triggers* disponíveis



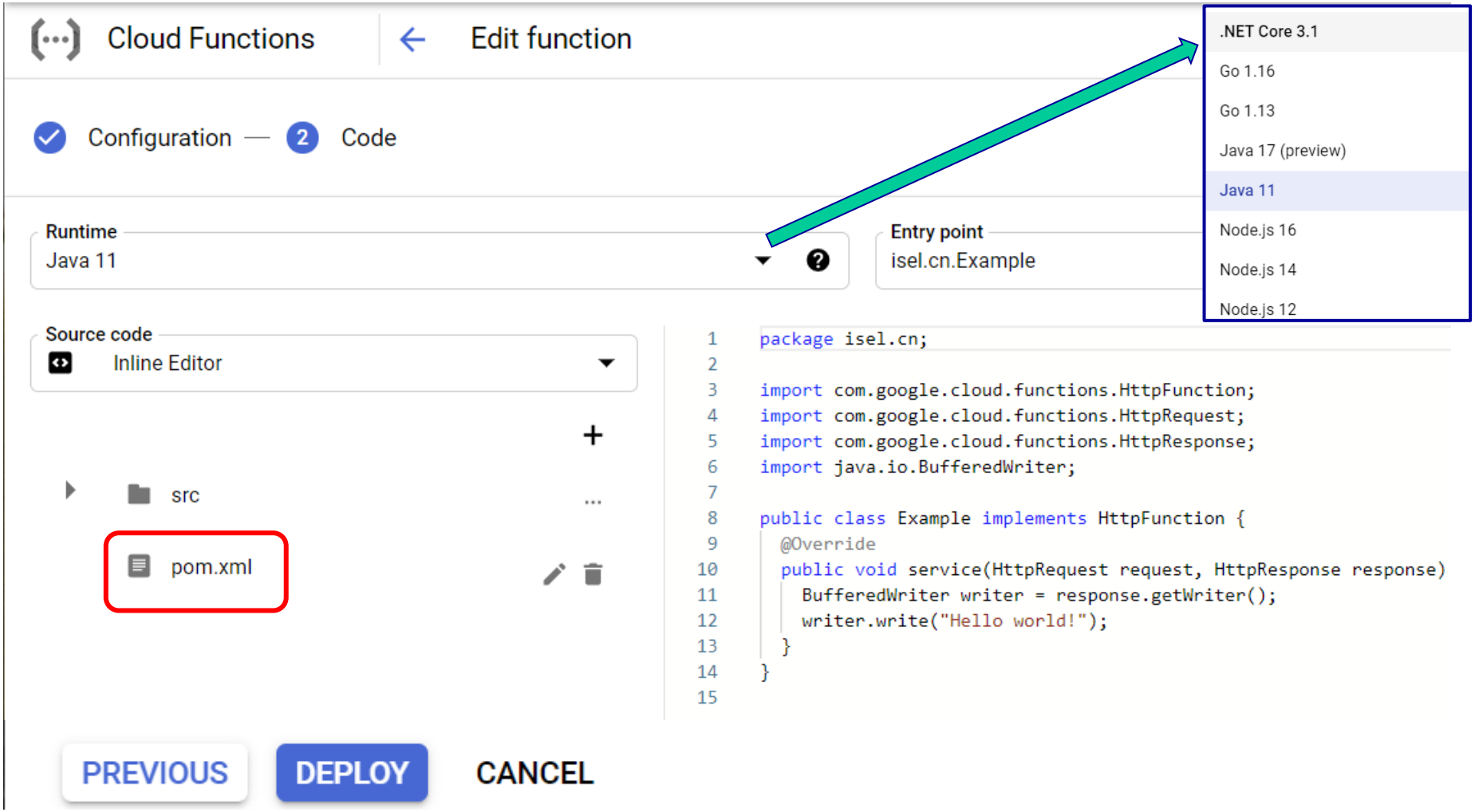
- A função poderá ter acesso público ou requerer autenticação (gerido no IAM)

Google Cloud Functions



- A memória alocada à função tem implicações na execução do código e no custo monetário de cada chamada
- Para aceder a serviços GCP é necessário associar à função uma conta de serviço com as permissões adequadas. Neste exemplo com acesso ao serviço *Storage*.
- É possível definir o número máximo de instâncias para limitar custos

Google Cloud Functions



https://europe-west1-cn2122-jsla-geral.cloudfunctions.net/cn-http-function

Devenvolvimento local

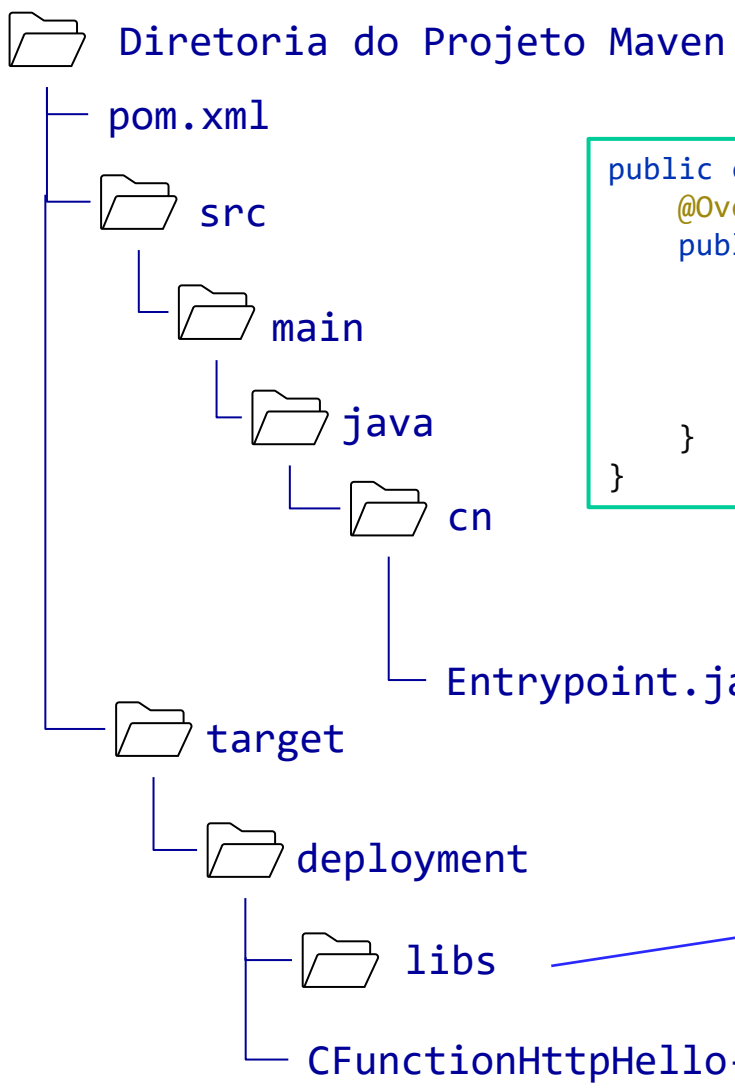
- Embora seja possível, é impraticável desenvolver *cloud functions* com código *inline*
- O desenvolvimento pode ser feito com projetos Maven locais, usando a estrutura de diretorias habitual, e as dependências necessárias: API *functions-framework* e outras dependências necessárias, por exemplo, de outros serviços GCP que a *function* utilize

```
<project …>
  <dependencies>
    <dependency>
      <groupId>com.google.cloud.functions</groupId>
      <artifactId>functions-framework-api</artifactId>
      <version>1.0.4</version>
    </dependency>
    ...
  <dependencies>
  ...
</project>
```

- Uma diretoria com os artefactos para *deployment* (slide seguinte)

Exemplo: CFunctionHttpHello.zip

Estrutura de um projeto maven




```java
public class Entrypoint implements HttpFunction {
    @Override
    public void service(HttpRequest request, HttpResponse response)
                                                        throws Exception {
        BufferedWriter writer = response.getWriter();
        String name=request.getFirstQueryParameter("name").orElse("World");
        writer.write("Hello "+name+" from Cloud Function http");
    }
}
```

Gerado automaticamente com *plugins* adequados

Maven *build: plugins* (1)

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.2.2</version>
      <configuration>
        <archive>
          <manifest>
            <addClasspath>true</addClasspath>
            <classpathPrefix>libs/</classpathPrefix>
          </manifest>
        </archive>
      </configuration>
    </plugin>
    ...
```

Geração do jar

Copiar dependências para diretoria *libs*

```
. . .
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>3.3.0</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <overWriteReleases>false</overWriteReleases>
        <includeScope>runtime</includeScope>
        <outputDirectory>${project.build.directory}/libs</outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Maven *build: plugins* (2)

Copiar *resources* para diretoria *libs*

```
...
<plugin>
     <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-resources-plugin</artifactId>
  <version>3.2.0</version>
  <executions>
    <execution>
      <id>copy-resources</id>
      <phase>package</phase>
      <goals><goal>copy-resources</goal></goals>
      <configuration>

<outputDirectory>${project.build.directory}/deployment</outputDirectory>
        <resources>
          <resource>
            <directory>${project.build.directory}</directory>
            <includes>
              <include>${project.build.finalName}.jar</include>
              <include>libs/**</include>
            </includes>
            <filtering>false</filtering>
          </resource>
        </resources>
      </configuration>
    </execution>
  </executions>
</plugin>

</plugins>
```

Deployment com linha de comandos

- Após fazer package do projeto Maven, utiliza-se a ferramenta `gcloud` para fazer *deployment* invocando a ferramenta a partir da diretoria base do projeto (onde está o pom.xml)
  - `gcloud functions deploy funcHttHello`
    `--allow-unauthenticated`
    `--entry-point=functionhttp.Entrypoint`
    `--runtime=java11`
    `--trigger-http`
    `--region=europe-west1`
    `--source=target/deployment`

Nome da função

Acesso à função sem autenticação

Nome da classe, ponto de entrada

Evento que desencadeia a execução da função

Região de *deployment*: `gcloud functions regions list`

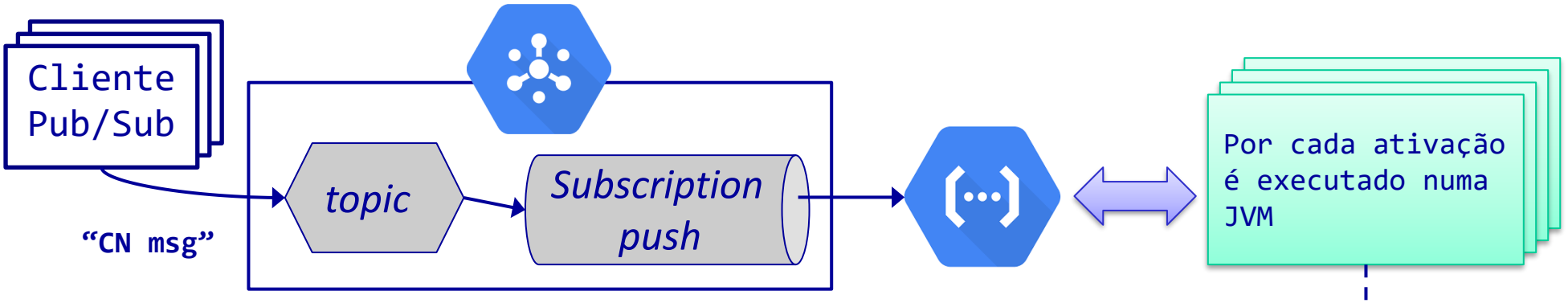
Diretoria com JAR e libs das dependências

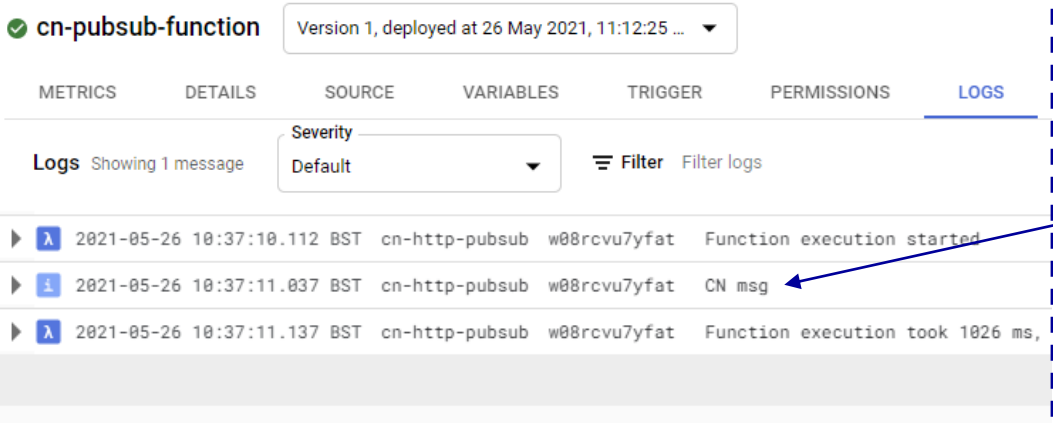
- Outros parâmetros relevantes:
  `--service-account=<service account>@project-id.iam.gserviceaccount.com`
  `--max-instances` controla o número máximo de instâncias que podem estar em execução (ex: `--max-instances=3`), útil para controlar custos e concorrência

Conta de serviço com permissões necessárias para acesso aos serviços GCP usados no código da *cloud function*

https://cloud.google.com/sdk/gcloud/reference/functions/deploy

Invocação com *Trigger* Pub/Sub






```
public class Example implements
                    BackgroundFunction<Message> {
  private static final Logger logger =
    Logger.getLogger(Example.class.getName());

  @Override
  public void accept(Message msg, Context context) {
    String data = new String(
              Base64.getDecoder().decode(msg.data));
    logger.info(data);
  }
}

public class Message {
    String data;
    Map<String, String> attributes;
}
```

Função *trigger* Pub/Sub – *Firestore (1)*

- Cada mensagem publicada um tópico a função é ativada e escreve a mensagem numa coleção no serviço Firestore

```
public class Entrypoint  implements BackgroundFunction<PSmessage> {
    private static Firestore db = initFirestore();

    private static Firestore initFirestore() {
      try {
        GoogleCredentials credentials = GoogleCredentials.getApplicationDefault();
        FirestoreOptions options = FirestoreOptions
                                        .newBuilder().setCredentials(credentials).build();
        Firestore db = options.getService();
        return db;
      } catch (IOException e) {
          e.printStackTrace();
      }
      return null;
    }

    @Override
    public void accept(PSmessage message, Context context) throws Exception {
       // ver slide seguinte
    }
}
```

```
public class PSmessage {
    String data;
    Map<String, String> attributes;
}
```

Exemplo: CFunctionPubSubFirestore.zip

Função trigger Pub/Sub – *Firestore (2)*

```
private static final Logger logger = Logger.getLogger(Entrypoint.class.getName());

@Override
public void accept(PSmessage message, Context context) throws Exception {
    if (db == null) {
        logger.info("Error connecting to Firestore. Exiting function.");
        throw new RuntimeException("Error connecting to Firestore");
    }
    logger.info("original message " + message.data);
    String dataAsString = new String(Base64.getDecoder().decode(message.data));
    logger.info(dataAsString);
    CollectionReference colRef = db.collection("CFPubSubMessages");
    // O message ID vem no eventID
    DocumentReference docRef = colRef.document(""+context.eventId());
    HashMap<String, Object> map = new HashMap<String, Object>();
    map.put("msg-data", message.data);
    map.put("data", dataAsString);
    if (data.compareTo("error") == 0)
       throw new Exception("error forced from data");
    map.put("ctx-messageId", context.eventId());
    map.put("ctx-pubTime", context.timestamp());
    ApiFuture<WriteResult> result = docRef.set(map);
    result.get();
    logger.info("Event was written to Firestore.");
}
```

O *messageId* e *timestamp* são passados no contexto

Deploy de função com *trigger PubSub*

```
gcloud functions deploy funcPubSub
  --region=europe-west1
  --entry-point=functionpubsub.Entrypoint
  --runtime=java11
  --trigger-topic cf-topic-pubsub-base
  --source=target/deployment
  --service-account=firestore@pjbase-cn2122.iam.gserviceaccount.com
```

Exemplo de conta de serviço com permissões para acesso ao Firestore

- O tópico é criado automaticamente no *deploy*
- A *subscription push* é criada automaticamente no *deploy*
- Sobre *acknowledge* das mensagens*:*
  - O *acknowledge* é implicitamente assumido pelo retorno normal do método *accept*. Se o método gerar uma falha a mensagem não será repetida.
  - Existe um mecanismo de configurar a função com propriedade `--retry` (*retry on failure*). No entanto, a propriedade de *retry* deve ser usada com cuidado pois pode conduzir a comportamentos de repetição de mensagens inadequados.

"If your function crashes for any reason, including bugs in your code, it will be retried for up to seven days if this box is checked. It is best used to make your function resilient to transient failures in your code."
https://cloud.google.com/functions/docs/bestpractices/retries#set_an_end_condition_to_avoid_infinite_retry_loops

Conclusões

- Vantagens
  - Programadores com foco no negócio e não em questões de infraestrutura
  - Adequadas para aplicações *event-driven*
  - Escalabilidade automática (elasticidade)
  - Redução de custos e redução do "*time to market*"
- Desvantagens
  - Não adequado para *long-term tasks*
  - O serem *Stateless* implica usar mecanismos externos para partilha de estado
  - Desadequadas para requisitos de baixa latência
  - Dificuldades de teste e *debugging*
  - Novos desafios de segurança
  - *Vendor lock-in*

Função *trigger* HTTP com compute

```
public class Entrypoint implements HttpFunction {

  @Override
  public void service(HttpRequest request, HttpResponse response) throws Exception {
    BufferedWriter writer = response.getWriter();
    String projectID = "cn2122-jsla-geral";
    String zone = request.getFirstQueryParameter("zone").orElse("europe-west1-b");
    writer.write("List running Vms in zone="+zone+"\n");
    try (InstancesClient client = InstancesClient.create()) {
      for (Instance instance : client.list(projectID, zone).iterateAll()) {
        if (instance.getStatus().compareTo("RUNNING") == 0) {
           writer.write("Name: " + instance.getName() +"\n");
           String ip = instance.getNetworkInterfaces(0).getAccessConfigs(0).getNatIP();
           writer.write("    Last Start time: " + instance.getLastStartTimestamp()+"\n");
           writer.write("    IP: " + ip+"\n");
        }
      }
    }
  }
}
```

- ✓ Colocar no pom.xml a dependência do serviço *Compute Engine*
- ✓ No *deploy* passar uma conta de serviço (`--service-account`) com permissões para acesso ao serviço, por exemplo, com *role Compute Admin*

Trabalho Final: Cloud Function *Lookup* de IP

```
public void service(HttpRequest request, HttpResponse response) throws Exception
{
    // 1. Obter o query parameter do request com nome do instance-group
    // dos servidores gRPC.
    // 2. Chamar a API do compute para obter a lista de ips dos servidores gRPC
    // 3. Retornar em response a lista de ips, por exemplo uma string com um
    // formato específico ou objeto JSON
}
```

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.9.0</version>
</dependency>
```

Exemplo de chamada de função Http com package **java.net.http**

```
String cfURL="https://us-central1-project-id.cloudfunctions.net/funcHttpHello?name=students";

HttpClient client = HttpClient.newBuilder().build();
HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(cfURL))
            .GET()
            .build();
HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
if(response.statusCode() == 200) System.out.println(response.body());
```