KotlinConf, Amsterdam
Oct 4, 2018

Full Stack Kotlin on Google Cloud

Brent Shaffer, Google Cloud DPE
bshaffer

Google Cloud

Brent Shaffer

Google Cloud

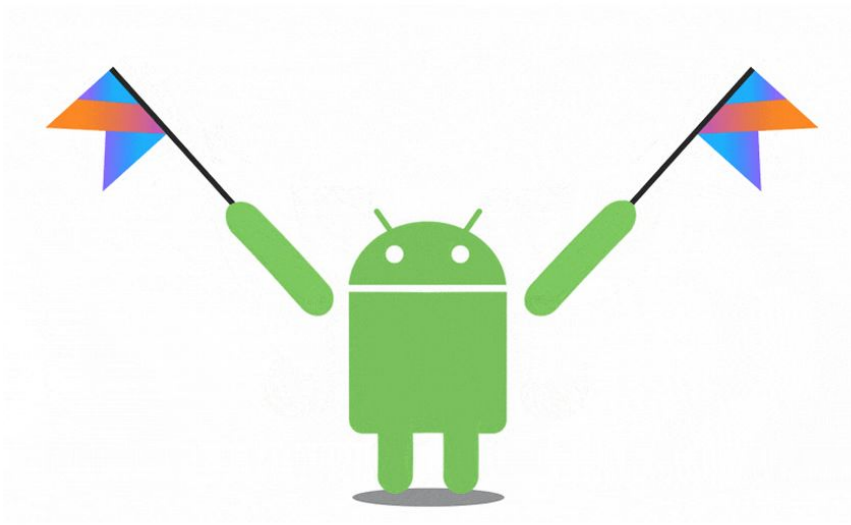
No language makes you feel more like a cog in a machine than Java

Google Employee
who will remain anonymous

Google Cloud

Life's better with Kotlin™!

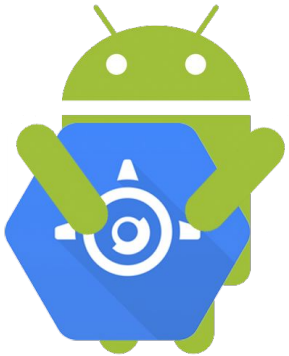
- Kotlin + Java interoperability
- Java++
- Instant improvement of Java Libraries
- Creative syntax
- Frontend & backend harmony
- Exciting growing community



If you are...

A Java backend dev thinking of switching to Kotlin

A Kotlin Android dev wanting to build apps server-side



Google Cloud

Java devs

Google Cloud

The Google Cloud Java client library in Kotlin

```
val vision = ImageAnnotatorClient.create() // Create an Image Annotator
val img = Image.newBuilder().setContent(imgProto).build()
val feat = Feature.newBuilder().setType(Type.LABEL_DETECTION).build() // Image builder
val req = AnnotateImageRequest.newBuilder()
    .addFeatures(feat)
    .setImage(img)
    .build()

requests.add(req)
```

A Kotlin-specific client library (using DSLs)

```
val request = annotateImageRequest { this: AnnotateImageRequest.Builder
    feature { this: Feature.Builder
        type = Feature.Type.LABEL_DETECTION
        maxResults = 2
    }
    image { this: Image.Builder
        content = ByteString.copyFrom(file.readBytes())
    }
}
requests.add(request)
```

Android devs

cloud.google.com/kotlin

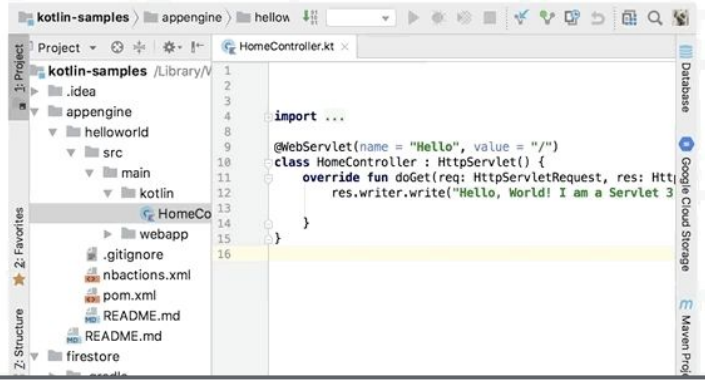
KOTLIN ON GOOGLE CLOUD PLATFORM

A community-supported project

QUICKSTART | COMMUNITY TUTORIALS

Build rich backends in your favorite language

- Create and deploy autoscaling Kotlin backends for your mobile apps
- Write Kotlin using your favorite Java frameworks and libraries
- Use fully-managed MySQL/PostgreSQL or Firebase for your application database
- Run Kotlin in a containerized application at scale on Google Kubernetes Engine



Google Cloud

KotlinConf, Amsterdam
Oct 4, 2018

Full Stack Kotlin on Google Cloud

Brent Shaffer, Google Cloud DPE
bshaffer

Google Cloud


FULL
STACK

- **Demo 1:** Extending Android apps with autoscaling backends

- **Demo 2:** Remotely configuring Android Things with IoT Core

Extending Android apps with autoscaling backends

g.co/codelabs/emojify

Google Cloud

Kotlin on GCP

fast deployment, infinite scalability!

- Compute Engine
- App Engine
- Kubernetes Engine

Spring Boot

- One of most popular Java frameworks for backend apps
- **Convention over Configuration**
- Kotlin support
- Get started right away with **start.spring.io**
- Runs great on App Engine
- Framework features like Dependency Injection, REST annotations, etc

Google App Engine

- One of the first if not THE first **serverless** offerings
- Developer provides code
- Google runs it at scale (0 to millions QPS)
- Now supports 7 languages

App Engine Elasticity

- $\lim_{req \to +\infty}$ **instances(req)** = $+\infty$
- $\lim_{req \to 0}$ **instances(req)** = 0
- Pay per use
- Consumes resources only when your code is running

Configuring auto-scaling

```
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
    <threadsafe>true</threadsafe>
    <runtime>java8</runtime>
    <system-properties>
        <property name="java.util.logging.config.file" value="WEB-INF/logging.properties"/>
    </system-properties>
    <automatic-scaling>
        <min-instances>1</min-instances>
        <max-instances>10</max-instances>
    </automatic-scaling>
</appengine-web-app>
```

Java 8 App Engine Standard Runtime

- **Without previous versions security limitations**
- Launched GA since last year
- Backward compatibility for millions of apps
- Open JDK 8 and Jetty 9 (Servlet 3.1 based)

gVisor

- **New App Engine Security Sandbox**
- Lightweight & fast
- Run any Java code!

The app: Emojify!

- Find faces
- Predict emotion
- Overlay emoji

Google Cloud

Google Cloud

Cloud Client Libraries

Cloud Vision (face detection, image annotation)

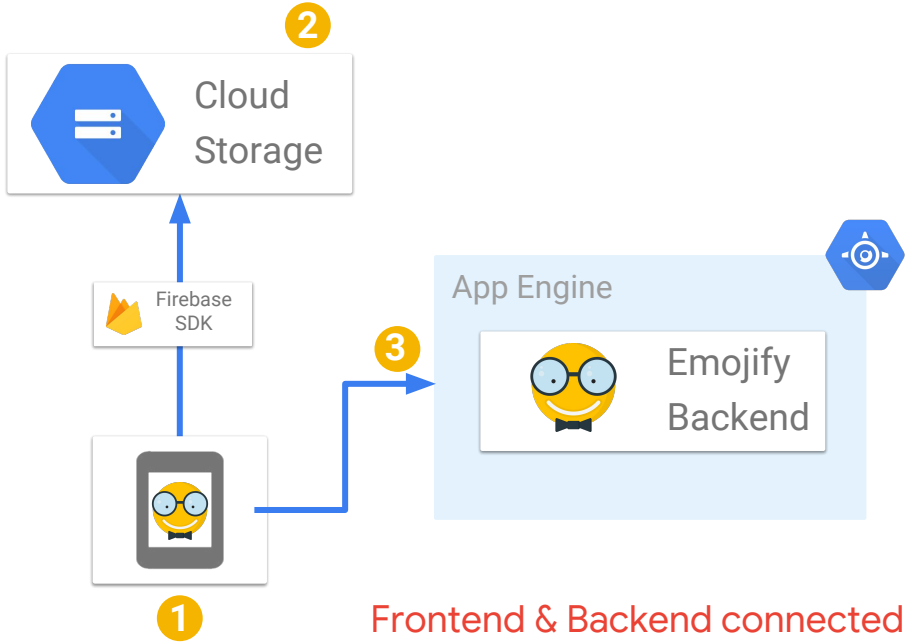
Cloud Storage (stores source and emojified images)

Frontend workflow

1 User opens App and snaps a picture

2 App uploads the image to Storage using the Firebase SDK

3 App makes an API call to the App Engine backend and supplies the image URI



Frontend & Backend connected to same Google Cloud Project!

Backend app architecture

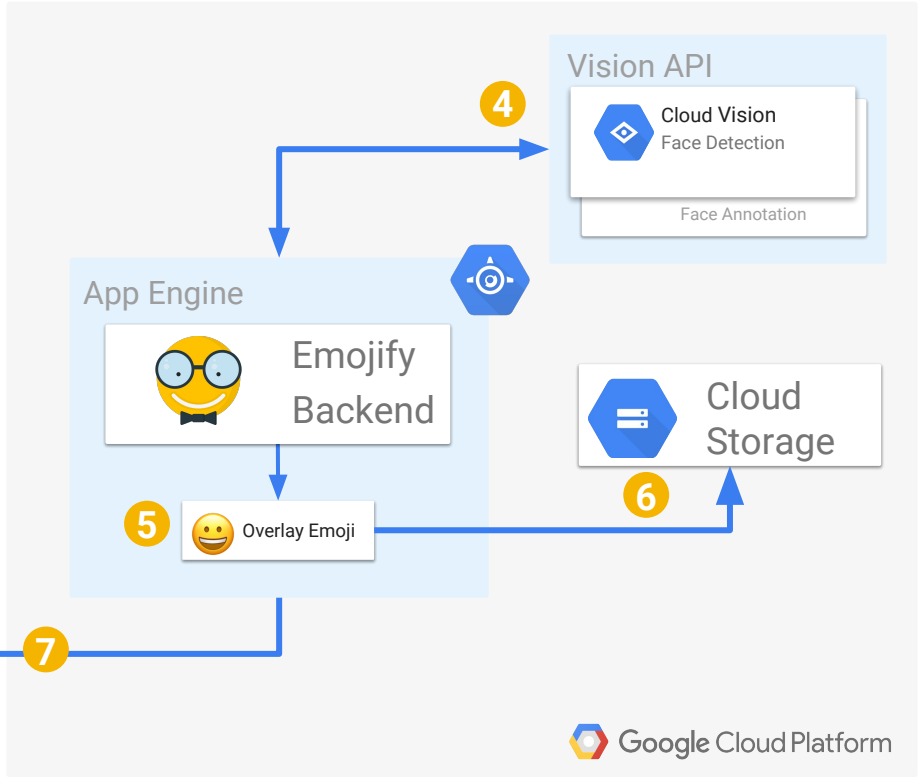
Backend workflow

4 Backend calls Vision API with the image URI.
The Vision API then returns the location of faces and their sentiment analysis

5 Backend overlays suitable emoji

6 Backend uploads emojified image to Storage

7 Backend returns emojified image URI to the App



Live selfies?

Uploading picture to Storage (Android)

```
private fun uploadImage(path: String) {
    val file = Uri.fromFile(File(path))
    imageId = "${System.currentTimeMillis()}.jpg"
    val imgRef = storageRef.child(imageId)
    updateUI {
        imageView.visibility = View.GONE
        tvMessage!!.text = "Crossing fingers…"
    }
    imgRef.putFile(file, StorageMetadata.Builder().setContentType("image/jpg").build())
            .addOnSuccessListener { _ ->
                updateUI { tvMessage.text = "Something is coming…" }
                callEmojifyBackend()
            }
            .addOnFailureListener { err ->
                updateUI {
                    show( msg: "Cloud Storage error!")
                    tvMessage.text = "There was an error with Cloud Storage!"
                }
                Log.e( tag: "storage", err.message)
            }
}
```

Google Cloud

Calling Emojify Backend (Android)

```
private fun callEmojifyBackend() {
    val queue = Volley.newRequestQueue( context: this)
    val url = "${this.backendUrl}/emojify?objectName=$imageId"
    updateUI { show( msg: "Image uploaded to Storage!") }
    val request = JsonObjectRequest(Request.Method.GET, url,  jsonRequest: null,
            Response.Listener { response ->
                val statusCode = response["statusCode"]
                /* Perform some checks and update UI */
            },
            Response.ErrorListener { err ->
                /* Log error message and update UI */
            })
    request.retryPolicy = DefaultRetryPolicy( initialTimeoutMs: 50000,  maxNumRetries: 5, DefaultRetryPolicy.DEFAULT_BACKOFF_MULT)
    queue.add(request)
}
```

REST Controller

```
@GetMapping( ...value: "/emojify")
fun emojify(@RequestParam(value = "objectName") objectName: String): EmojifyResponse {

    if (objectName.isEmpty()) return errorResponse(HttpStatus.BAD_REQUEST, errorCode: 106)

    if (objectName.contains( char: '/')) return errorResponse(HttpStatus.BAD_REQUEST, errorCode: 101)

    val bucket = storage.get(bucketName) ?: return errorResponse(HttpStatus.INTERNAL_SERVER_ERROR, errorCode: 102)
    val publicUrl: String =
        "https://storage.googleapis.com/$bucketName/emojified/emojified-$objectName" // api response
```

Google Cloud

Calling Vision (Backend)

```
// Setting up image annotation request
val source = ImageSource.newBuilder().setGcsImageUri("gs://$bucketName/$objectName").build()
val img = Image.newBuilder().setSource(source).build()
val feat = Feature.newBuilder().setType(Type.FACE_DETECTION).build()
val request = AnnotateImageRequest.newBuilder()
    .addFeatures(feat)
    .setImage(img)
    .build()

// Calls vision api on above image annotation requests
val response = vision.batchAnnotateImages(listOf(request))
```

Drawing Emoji (Backend)

```
// Writing source image to InputStream
val imgBuff = stream(objectName)
val gfx = imgBuff.createGraphics()

if (resp.faceAnnotationsList.size == 0) return errorResponse(HttpStatus.BAD_REQUEST,  errorCode: 107)

for (annotation in resp.faceAnnotationsList) {
    val imgEmoji = emojiBufferedImage[bestEmoji(annotation)]
    val poly = Polygon()
    for (vertex in annotation.fdBoundingPoly.verticesList) {
        poly.addPoint(vertex.x, vertex.y)
    }
    val height = poly.ypoints[2] - poly.ypoints[0]
    val width = poly.xpoints[1] - poly.xpoints[0]
    // Draws emoji on detected face
    gfx.drawImage(imgEmoji, poly.xpoints[0], poly.ypoints[1], height, width,  observer: null)
}
```

Google Cloud

Uploading result to Storage (Backend)

```
// Writing emojified image to OutputStream
val outputStream = ByteArrayOutputStream()
ImageIO.write(imgBuff, imgType, outputStream)

// Uploading emojified image to GCS and making it public
bucket.create(
    "emojified/emojified-$objectName",
    outputStream.toByteArray(),
    Bucket.BlobTargetOption.predefinedAcl(Storage.PredefinedAcl.PUBLIC_READ)
)
```

Google Cloud + Emojify

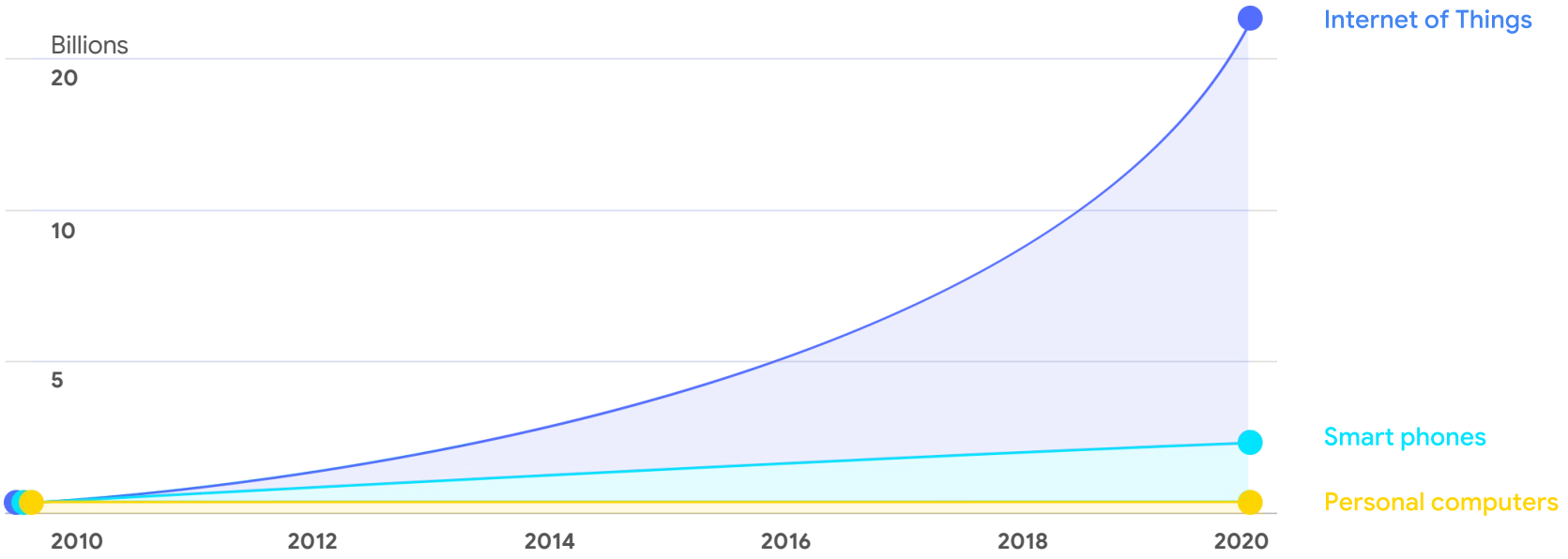
- **Firebase SDK** on Android
- **Cloud Storage** and **Cloud Vision** client library on Backend
- All running on **App Engine**

Remotely configuring Android Things with IoT Core

bit.ly/iot-core-codelab

Google Cloud

Significant growth in IoT (still) coming



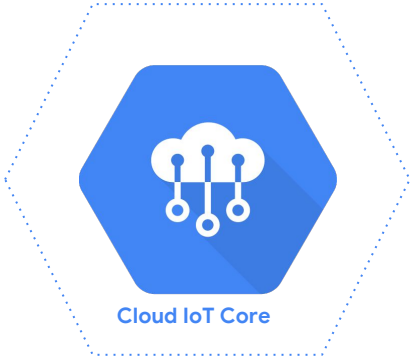
Google Cloud

Source: “The internet of everything 2015”, Business Insider Intelligence modified with information from a later Gardner 2017 report

Cloud IoT Core: Fully-managed service to securely connect and manage your global device network

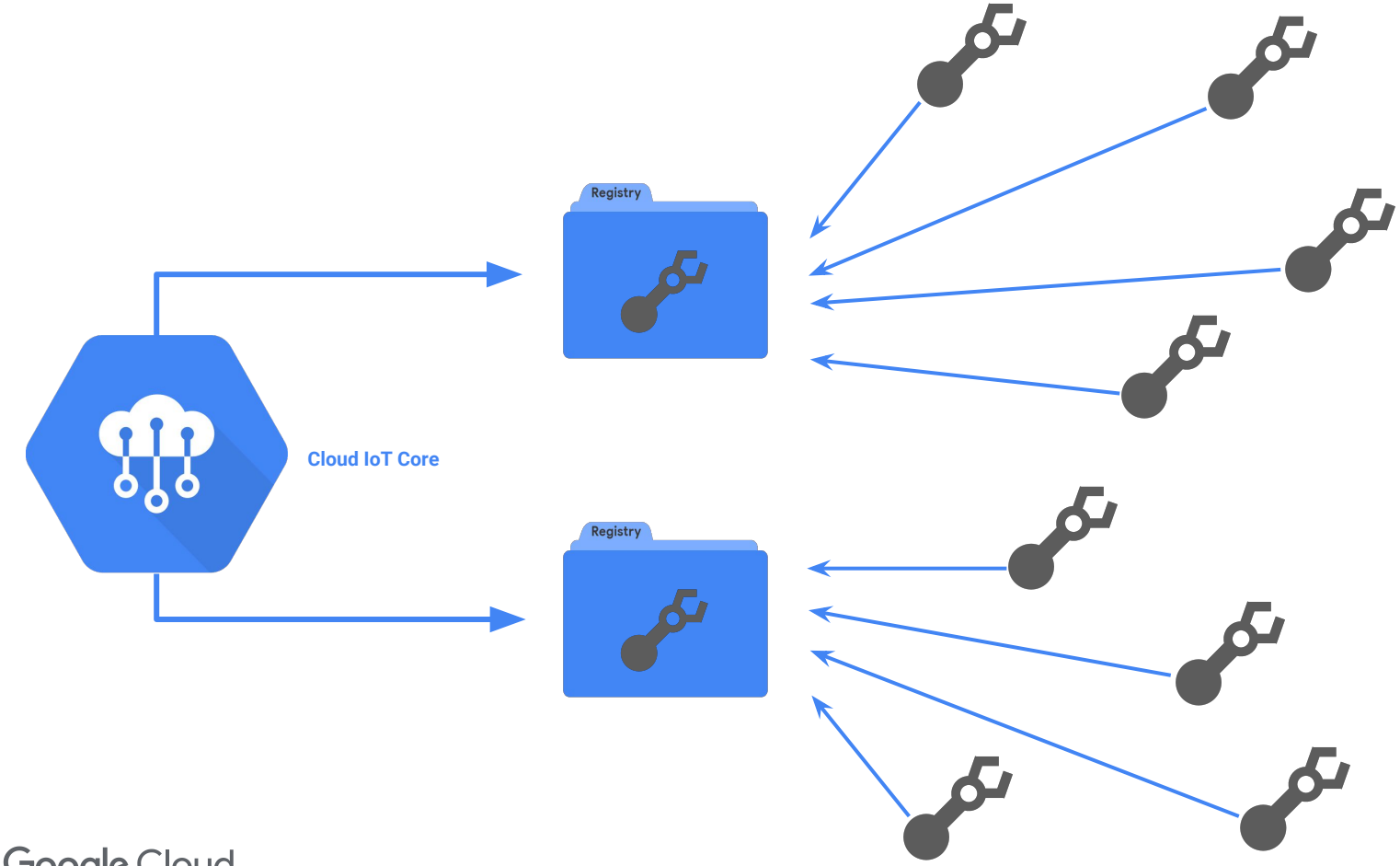
Protocol Bridge :

- **Protocol** (Https/MQTT) endpoint
- Automatic load balancing
- **Global data access** with Pub/Sub

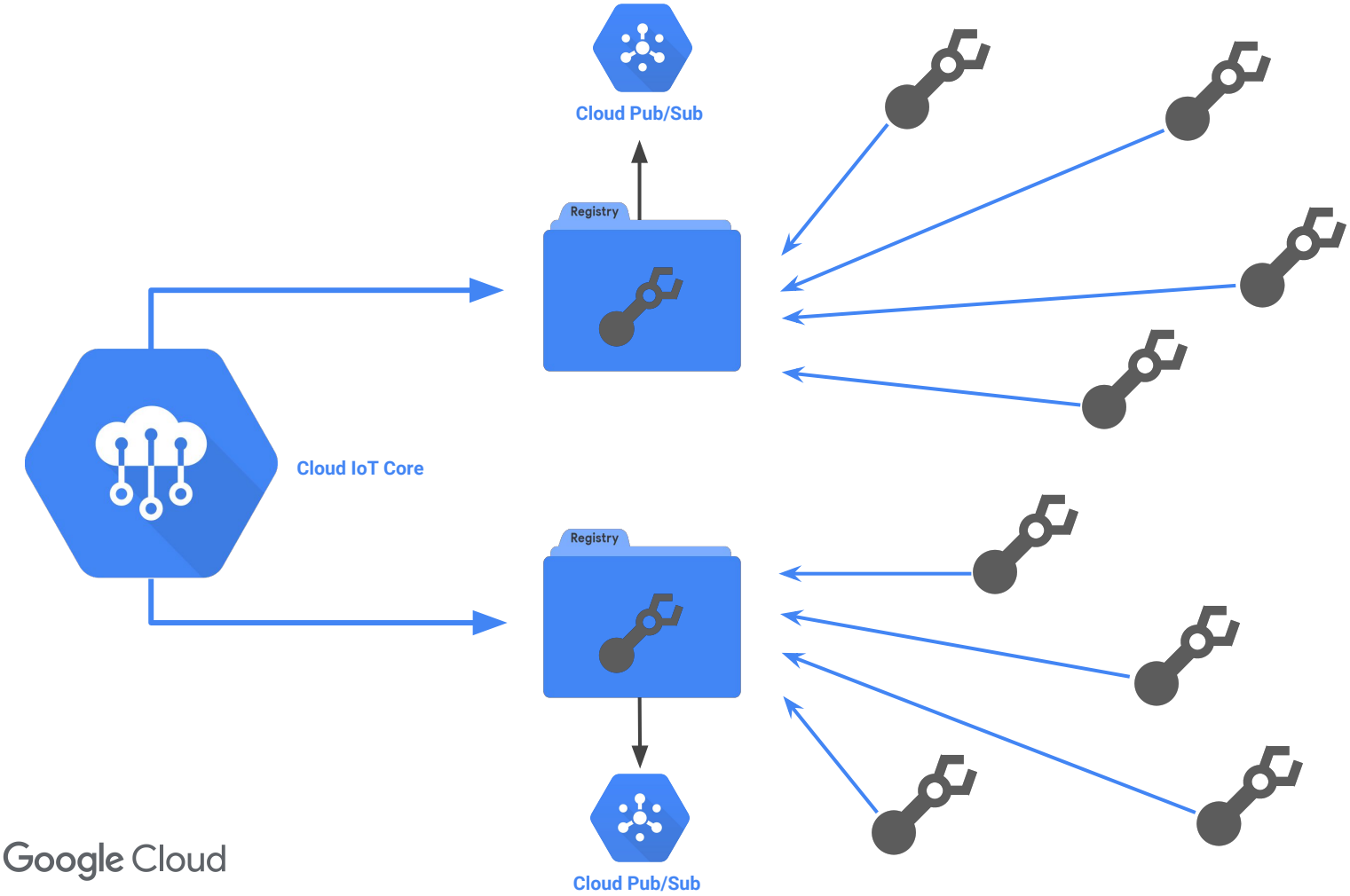


Device Manager :

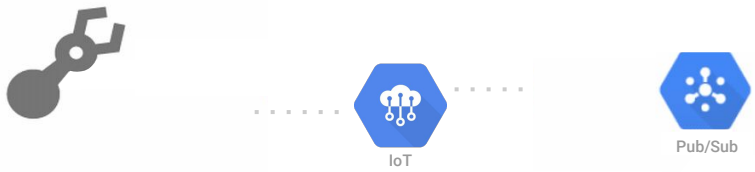
- Configure individual devices
- Update and **control devices**
- Role level access control
- Console and APIs for device deployment and monitoring

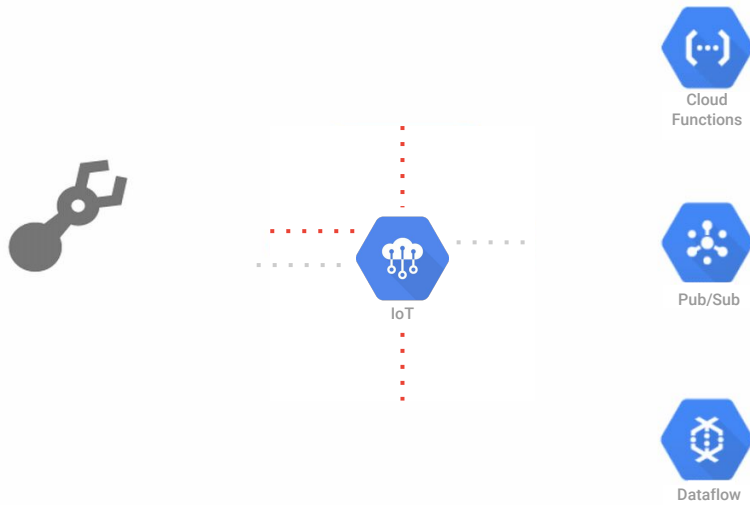


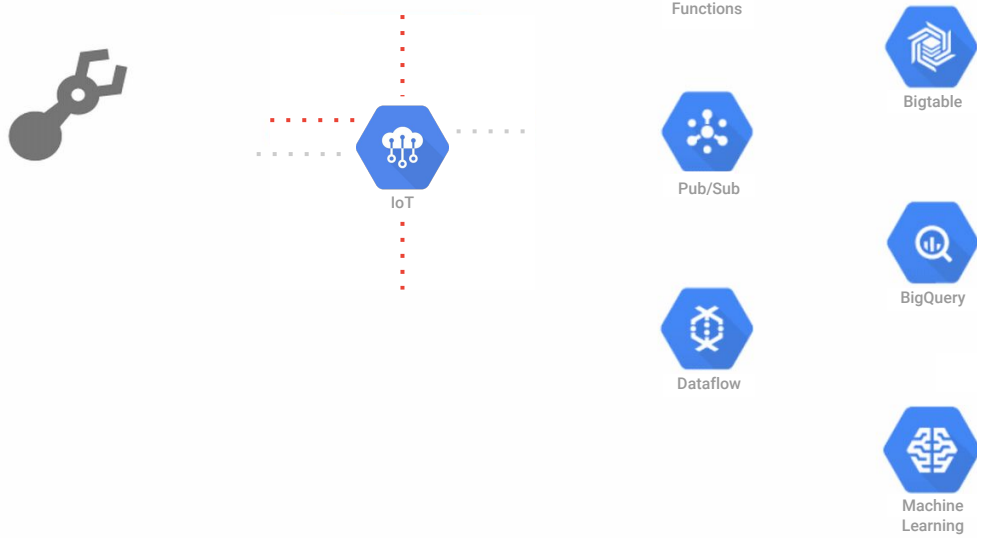
Google Cloud

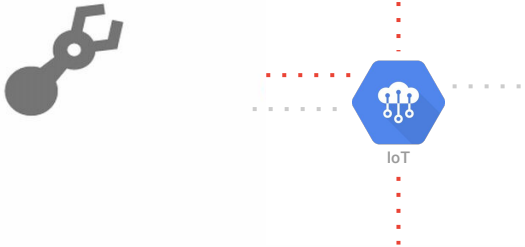


Google Cloud


IoT
Pub/Sub


IoT
Cloud Functions
Pub/Sub
Dataflow


IoT
Cloud Functions
Pub/Sub
Dataflow
Bigtable
BigQuery
Machine Learning



Cloud Functions

Pub/Sub

Dataflow

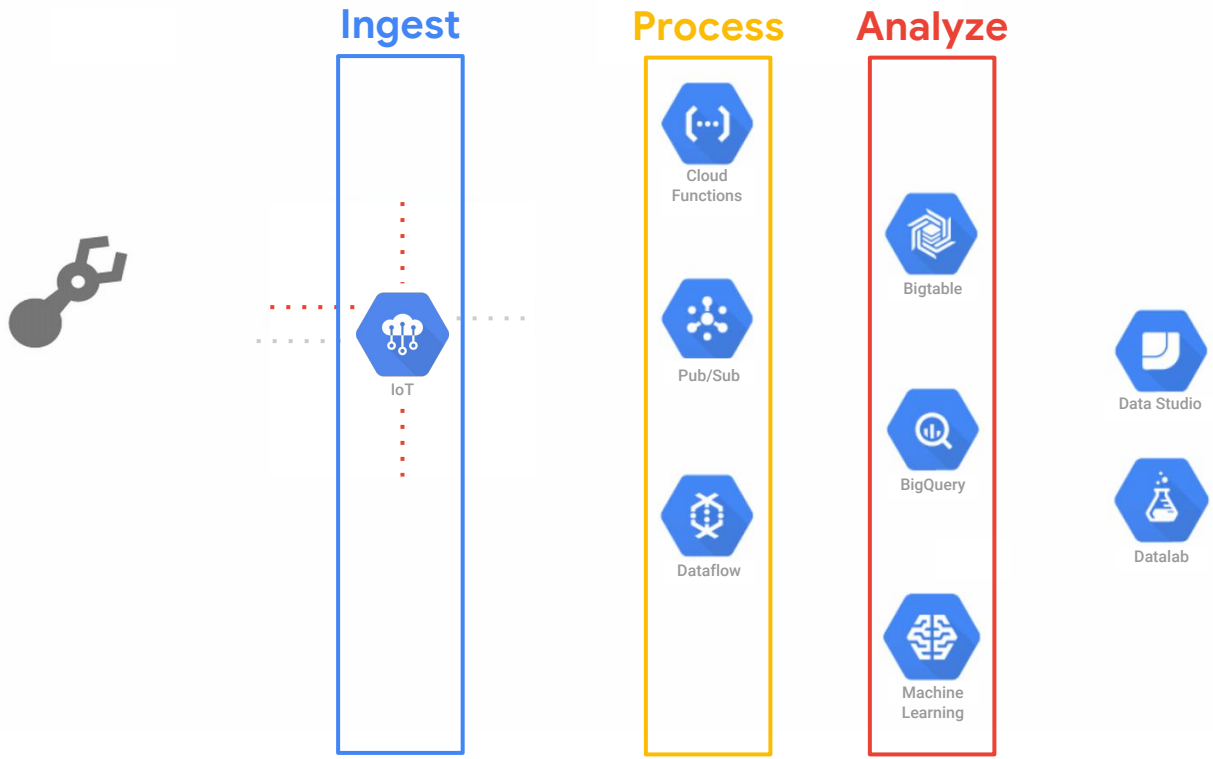
Bigtable

BigQuery

Machine Learning

Data Studio

Datalab


Ingest
Process
Analyze
IoT
Cloud Functions
Pub/Sub
Dataflow
Bigtable
BigQuery
Machine Learning
Data Studio
Datalab

The app: Coffee Heater!

- **Raspberry Pi** controls the device
- **Rainbow Hat** indicates the temperature and the heater's state

androidthings

Google Cloud

The app: Coffee Heater!

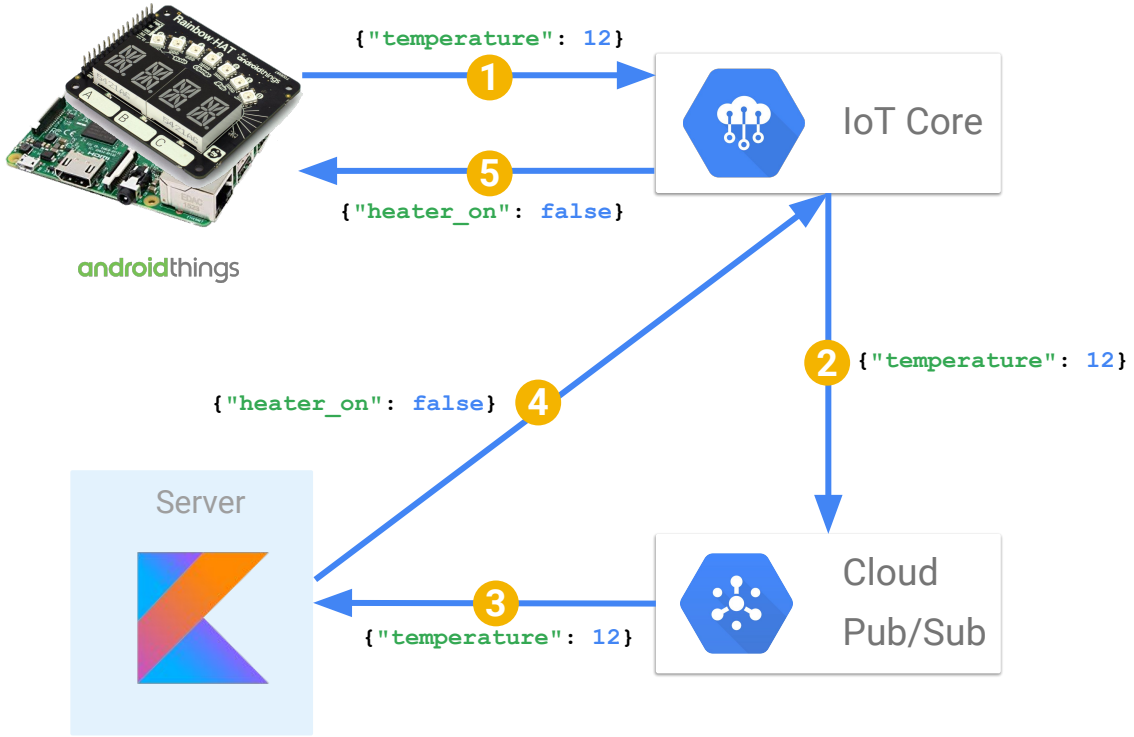
- **Raspberry Pi** controls the device
- **Rainbow Hat** indicates the temperature and the heater's state
- **Kotlin server** listens to Cloud Pub/Sub for the temperature of the coffee and modifies the device configuration to toggle the heater

Google Cloud

IoT Device Demo Architecture

IoT Core workflow

1. Device publishes data on MQTT topic
2. Config and metadata is published to Cloud Pub/Sub on device registry's topic
3. Server receives telemetry and metadata from cloud Pub/Sub
4. Server updates device config in Cloud IoT
5. Device receives new config on MQTT topic



Google Cloud

Rainbow Time

Google Cloud

Create IoT client and set a listener (device)

```
// Initialize the IoT Core client
client = IotCoreClient.Builder()
        .setConnectionParams(connectionParams)
        .setKeyPair(keys)
        .setOnConfigurationListener(OnConfigurationListener {
            this.onConfigurationReceived(it)
        })
        .build()

// Connect to Cloud IoT Core
client.connect()
```

Google Cloud

Publish telemetry data (device)

```
Log.d(TAG, "Publishing telemetry event")

val payload = JsonObject(mapOf("temperature" to currTemp.toInt()))
    .toJsonString()
val event = TelemetryEvent(
    payload.toByteArray(), null, TelemetryEvent.QOS_AT_LEAST_ONCE)

client.publishTelemetry(event)
```

Listen to telemetry events using PubSub (server)

```
println("Message Id: ${message.messageId} Data: ${message.data.toStringUtf8()}")

val data = Klaxon().parse<IotData>(message.data.toStringUtf8())
    ?: return println("Loading JSON payload failed.")

// Get the registry id and device id from the attributes. These are
// automatically supplied by IoT, and allow the server to determine which
// device sent the event.
val deviceId = message.attributesMap["deviceId"]
val deviceName = DeviceName.format(
    message.attributesMap["projectId"],
    message.attributesMap["deviceRegistryLocation"],
    message.attributesMap["deviceRegistryId"],
    deviceId
)

println("The device ($deviceId) has a temperature of ${data.temperature}")
```

Google Cloud

Modify the device config to change state (server)

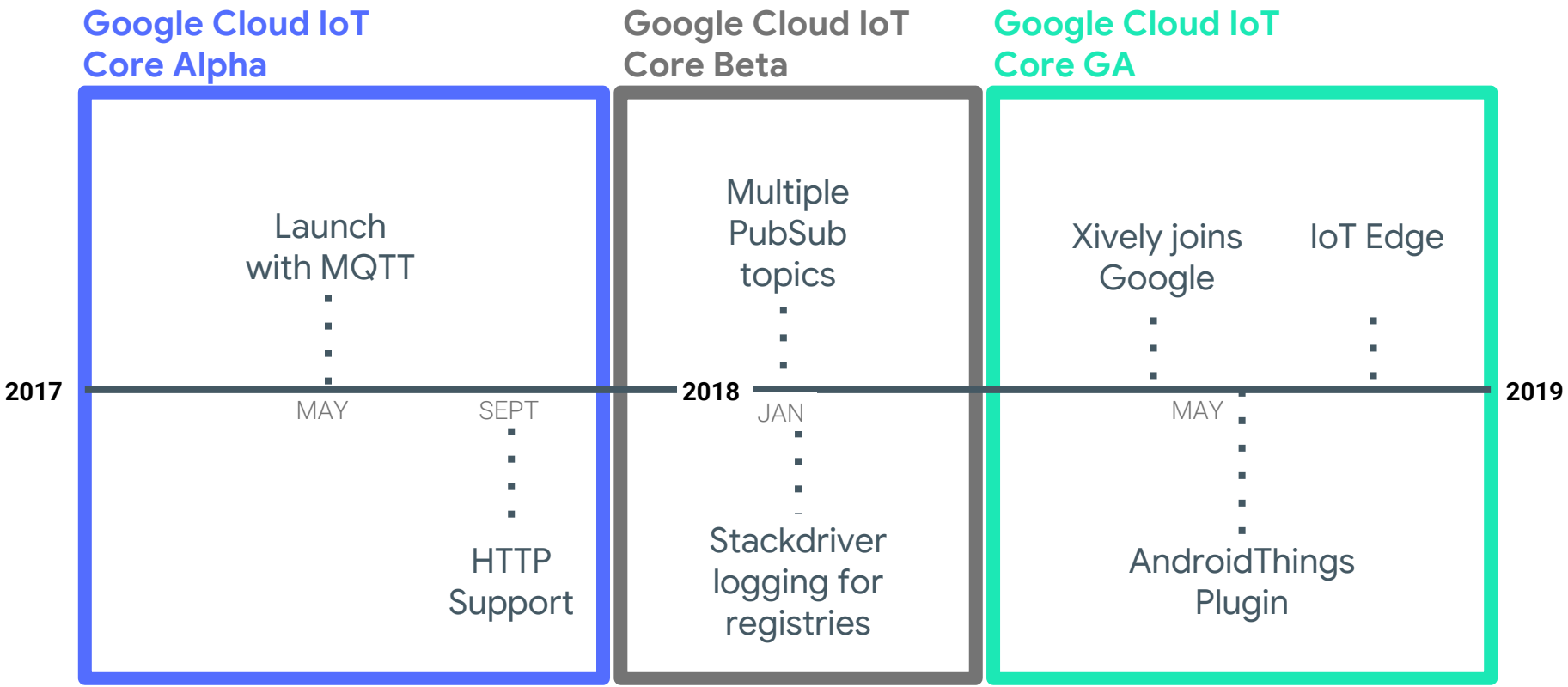
```
val json = JsonObject(mapOf("heater_on" to heaterOn)).toJsonString()

val req = ModifyCloudToDeviceConfigRequest.newBuilder()
    .setName(deviceName)
    .setVersionToUpdate(0)
    .setBinaryData(ByteString.copyFromUtf8(json))
    .build()

iot.modifyCloudToDeviceConfig(req)
```

Update device when configuration changes (device)

```
private fun onConfigurationReceived(bytes: ByteArray) {
    try {
        val config = Klaxon().parse<CoffeeControlConfig>(bytes.toString(UTF_8))
        if (config != null) {
            heaterOn = config.heaterOn
            Log.d(TAG, "Config: ${bytes.toString(UTF_8)}")
        } else {
            Log.d(TAG, "Invalid JSON string")
        }
    } catch (ke: KlaxonException) {
        Log.d(TAG, "Could not decode JSON body for config", ke)
    }
}
```


Google Cloud IoT Core Alpha
Google Cloud IoT Core Beta
Google Cloud IoT Core GA
Launch with MQTT
Multiple PubSub topics
Xively joins Google
IoT Edge
2017
MAY
SEPT
2018
JAN
MAY
2019
HTTP Support
Stackdriver logging for registries
AndroidThings Plugin

Thanks!
Questions?

Google Cloud