Serverless Application Architecture
Welcome to the Workshop

We will explore the ways that designing serverless applications differs from traditional server-side apps.

- **Serverless**
  - Of course there are still servers
  - But they aren't your problem

- The cloud provider handles scaling, deployment, monitoring, availability, backup

- Think of this as **managed** application platforms
I'm Charlie Engelke

Google Cloud Developer Relations Engineer
Specializing in serverless products

My career prior to Google was with a medium-sized application company
  Pretty much every tech job over time
  Led application development projects on lots of platforms

Cloud computing was a great equalizer, allowing us to compete against enormous competitors

My job now is to help developers take advantage of Google's offerings to solve their own problems
And what about Python?

- This isn't a workshop about Python, it's a workshop that uses Python

- Python is a core technology at Google
  - Google's first serverless offering, App Engine, was built with Python, for Python
  - (Guido van Rossum was involved)

- All our cloud offerings support Python

- I learned Python so I could use App Engine
  - Python and PyCon made me a Googler
Agenda

1. Serverless application characteristics

2. High-level design concepts for a very minimal serverless application

3. Description of a larger, distributed, serverless application and design
   - Slides, GitHub repo, codelabs, running demo, and videos at serverlessworkshop.dev

4. Q&A during the scheduled time in chat
Some Serverless differences

Serverless code runs **on-demand**
- When there's no work to do, it goes away

Serverless code is **stateless**
- When the code goes away, so does its memory and file system

Serverless code **scales** on demand
- Demand increases, more instances are provided
- Or it can scale down to zero
Details

- Most applications need state
  - So must use external data storage
  - Preferably serverless itself

- Events (like web requests) are handled by the platform
  - Which then invokes your code
  - Once the event is handled, code stops

- The application is no longer a program
  - It's a cooperating collection of pieces
Serverless Options
Important note:

- I am showing Google Cloud serverless offerings
  - I know them best, and after all, this is a sponsored workshop

- But every major cloud provider has similar offerings
  - Everything here can apply to most of them
  - Specifics will change, but concepts remain
  - I was a frequent, successful user of one of those cloud providers at my prior job
Google Cloud Serverless Compute

App Engine

Google's first serverless offering

- Launched in 2008, just became a teenager
- Python only at first, Java next, now 6 languages
- Useful as a web app backend

First generation included bundled APIs
- Current generation uses APIs available to all platforms instead
Google Cloud Serverless Compute

App Engine

Cloud Functions
- Good fit for small, focused event handlers
  - E.g., a web app user uploads an image that needs post processing
  - Instead of waiting for main app, have the upload trigger an event
  - A Cloud function handles the event
Google Cloud Serverless Compute

- App Engine
- Cloud Functions
- Cloud Run
  - Container based
    - Similar to App Engine and Cloud Functions, but takes a container
  - No longer limited to supported languages
Key differences

- **App Engine**
  - Bring your own source code, use provided run-time

- **Cloud Functions**

- **Cloud Run**
  - Bring your own container
Cloud Functions: short path to live code online

```python
def hello_world(request):
    """Responds to any HTTP request.

    Args:
    request (flask.Request): HTTP request object.

    Returns:
    This response text or any set of values that can be turned into a
    Response object using
    `make_response <http://flask.pocoo.org/docs/1.0/api/#flask.Flask.make_response>`_.
    
    .. code-block::

        request_json = request.get_json()
        if request.args and 'message' in request.args:
            return request.args.get('message')
        elif request_json and 'message' in request_json:
            return request_json['message']
        else:
            return f'Hello World!'
```
Google Cloud Serverless state (data)

- Cloud Firestore
- Firestore in Datastore Mode
- Cloud Storage

Document oriented NoSql

Blob store
Some Cloud Event Sources

- Web requests
- Updated data
- Tasks, Scheduler, Pub/Sub
Basic Serverless App Example
Consider a basic Todo app

- Keep it super simple
- One user (or a group sharing todo items)
- One list of items
- Any user can list, view, update, add, or delete items
One possible approach

- Get a Linux virtual machine
- Install a web server (e.g., NGINX)
- Install a programming language
  - Python, of course
- Install libraries
- Install a database server (MySQL? SQLite? Postgres?)
- Put in your source code
- Configure and start everything up
- Figure out backup, redundancy, disaster recovery, monitoring...
Or a serverless approach

- **State (persistent data)**
  - A list of items "to do"

- **Events**
  - Request to display the list
  - Request to add an item
  - Request to remove an item

- **Compute**
  - Respond to these requests by fetching from the list or modifying the list, as needed
Select tool(s) for State

- State
  - Cloud Firestore
  - Cloud Datastore ✔
  - Cloud Storage

Cloud Firestore would be a good fit, too.
Cloud Storage would not be as good for this use case.
Events

- Make each event a web request
  - GET / displays the list
  - POST / adds an item
  - DELETE /item_id deletes an item
    - or POST to /?
  - PUT /item_id updates an item
Serverless tools for compute

- Respond to web requests, update datastore
- Three strong serverless options
  - App Engine, the original
    - Cloud Functions, functions as a service
    - Cloud Run, serverless containers

Any of them can handle this well
App Engine has a little better fit, and can use Identity-Aware Proxy to handle user authentication
Identity-Aware Proxy

- Intercepts requests to your app
- Allows only authorized users through
  - List of email addresses
  - Google Groups
  - All email addresses in a Workspace domain
  - allUsers and allAuthenticatedUsers
- Adds headers to every request with user ID
- Easy to set up for App Engine, possible for other compute platforms via load balancers
Larger Serverless App
Student Programming Contest

serverlessworkshop.dev

- Faculty judges create problems to code
  - "read an input file, produce an output file"

- Students are given the problem descriptions and code solutions

- Solutions are turned in
  - Judges compile and run solutions with multiple data sets

- Students are told whether they passed, failed, timed out, or crashed
People involved

- **Judges**
  - Write up problems to solve with code
  - Check student solutions

- **Students**
  - Write a program for each problem
  - Create and use test data
  - Submit solutions for scoring

- **Managers**
  - Distribute problems to students
  - Accept solutions, assign judges
  - Track results
Solve with a distributed application

Each of the three parties has their own platform

- **Students** deploy solutions to their own server
- **Judges** run their tests against solutions
- **Managers** provide a web site
  - Form for accepting submitted solution URL
  - Create event triggering judging
  - Handle results from judges
  - Track and display standings
Projects

- All Google Cloud resources live in **projects**
  - Resources in the same project can usually interact with each other
  - You can enable resources in different projects to interact with each other

- Students, judges, and the manager each own their own separate projects
  - So they may need to allow the other projects to interact with them
  - The codelabs all use the same project to avoid this complexity to keep them simple
Is this reasonable?
Yes, with Cloud Functions
Example: play a game

- Student writes a solution
  - Accepts request representing game state
  - Responds with game move

- Deploys program to the internet
  - Submits the URL for judging

- Judging system makes web request with data in the body, solution returns output in response
1. Judge sends initial game state to player
2. Game player returns a move
3. Judge updates the game state, adding player's move and judge's next move in response
4. Repeat with updated game state
Overall system
- **Student**
  - Create solution, deploy to web
  - Submit URL via web form for judging

- **Judge**
  - Create judging program(s)
    - Trigger on new message to a topic
    - Exercise solution via web
  - Report result to manager

- **Manager**
  - Accept submissions
  - Publish message
  - Accept results from judges
  - Display web page
Solution platform

- Compute is Cloud Functions
  - Fewest steps to deploy
  - Creates a public URL
  - Function should allow requests from anyone

- Events
  - One web request provides the input, and the response has the output

- There is no state
  - If judging system wants a multi-step process, it includes the prior step results in the request
Judger platform

- Compute: Cloud Functions or Cloud Run
  - Interact with student submissions via sequence of web request/responses

- Events
  - Trigger on Pub/Sub message from manager
  - Report results to URL in message

- There is no state
  - Judger handles one message, may make series of requests to player, sends results, and it's done
Question: Cloud Function or Cloud Run?

- Cloud Functions are easier
  - Work with console, provide source code

- But Cloud Run is more flexible
  - Use containers
  - Not limited to specific runtimes

- Judges are faculty, and faculty can be idiosyncratic
  - "I need my solution in Rust/Cobol/obsolete version of otherwise supported language."
  - "I have an executable file I'll need to use."
Manager platform

- **Compute**
  - App Engine for website
    - Authenticate with Identity-Aware Proxy
  - Cloud Function to accept judges results

- **Events**
  - Web requests from people for App Engine
  - Web service request from judgers

- **State: Cloud Firestore**
  - Record submission
  - Add results to a subcollection
Overall coupling

Identity-Aware Proxy

Manager Web App

Database

Manager Service

Pub/Sub Topic

Judger

Internet

Student Laptop

Student Solution
Permissions

- Manager must allow-list students in IAP
- Manager can read/write to Firestore database and publish to Pub/Sub topic
- Manager must allow judges to subscribe to topic
- Manager function accepts HTTP submissions from judgers (authentication not required)
- Players accept HTTP requests from anyone
Permissions

- Manager must allow-list students in IAP
- Manager can read/write to Firestore database and publish to Pub/Sub topic
- **Manager must allow judges to subscribe to topic**
- Manager function accepts HTTP submissions from judgers (authentication not required)
- Players accept HTTP requests from anyone
Contest problem

- Play the simplest possible game: **guess a number**
  - Given minimum and maximum, and history of guesses
  - Respond with a whole number guess

- Don’t worry about the judging system for now (we're the student who has to write a player)

- You can try it out and submit your solution to example judging system
  [https://serverlessworkshopdemo.appspot.com/](https://serverlessworkshopdemo.appspot.com/)
Starting input example

{
   "minimum": 1,
   "maximum": 10,
   "history": []
}
Example output

6

Yes, this is the JSON representation of a whole number
Second example move request

```
{
    "minimum": 1,
    "maximum": 10,
    "history": [
        {
            "guess": 6, "result": "higher"
        }
    ]
}
```
Hands-on codelabs at

https://serverlessworkshop.dev
Wrap-up
Serverless Technologies Used

- Functions as a service (Cloud Functions/Run)
  - Student solution
  - Manager web service
  - Judgers (may prefer Cloud Run)

- Platform as a service (App Engine)

- Reliable messaging (Pub/Sub)

- NoSQL database (Firestore)

- User auth as a service (Identity-Aware Proxy)
Serverless application design: TL;DR

- Identify the data that must be maintained
- Note the events that can change that state
- Specify compute needed for each event
- Choose appropriate platforms for each
- Build each part as independently as possible
  - Should be possible to test each part without the rest of the system
Thank you! Questions via chat.

Visit https://severlessworkshop.dev/
Serverless Application Architecture