

Welcome to the workshop!

Serverless Architecture by Example

- Open up your laptop, connect to wifi, and open your (recent web) browser
- Log in to the Google Cloud console:
<https://console.cloud.google.com/>
- Set up a new Gmail account if needed



Serverless Architecture by Example

Strange Loop
St. Louis, MO
September 12, 2019

Laurie White (lauriewhite@google.com)
Charles Engelke (engelke@google.com)

Google Cloud

Google Cloud Developer Relations

Slides and exercises online at

<https://serverlessworkshop.dev>

Welcome to the workshop!

You will build a loosely-coupled, event driven, distributed serverless system today

So get your laptops ready!

But first, let's cover some background:

Just what is **serverless**, anyway?

The problem we want to solve

The tools we will be using

Serverless computing

Spoiler alert! *There are still servers. Don't tell anybody!*

- But they are the cloud platform's problem, not yours
- You don't have to provision, manage, monitor, or scale them
- And many serverless options scale down to zero when idle

There are different flavors of serverless computing

- Container based - the platform handles the kernel and scaling, you handle support systems (like libraries)
- Managed - you bring your application code, the platform handles everything else

This workshop uses managed serverless

You are responsible for your application code

- The platform handles all supporting software, monitoring platform health, and scaling
- Important - many platforms can scale to zero
 - So idle times don't have any compute costs

Your code may be unloaded, reloaded or loaded into multiple hosts at any time

- So you can't save any state in memory or on disk
- And you may have startup latency at times

Common characteristics of serverless

Stateless software

- External data stores are used when needed

Many pieces, loosely coupled

- Handle one task, trigger other pieces as needed for more

Event-driven

- Code runs when something happens
- A web request, a storage event, a message delivered

Asynchronous communications

- Send requests but don't wait for responses

The Problem: Programming Contests

- Participants are given a set of problems to code
 - In the form "read an input file, produce an output file"
- Contestants code solutions, test with provided sample data and (we hope) their own test data
- Solutions are turned in (physical media, email, etc.)
 - Judges compile and test solutions with multiple data sets
- Contestants are told whether they passed, failed, timed out, or crashed

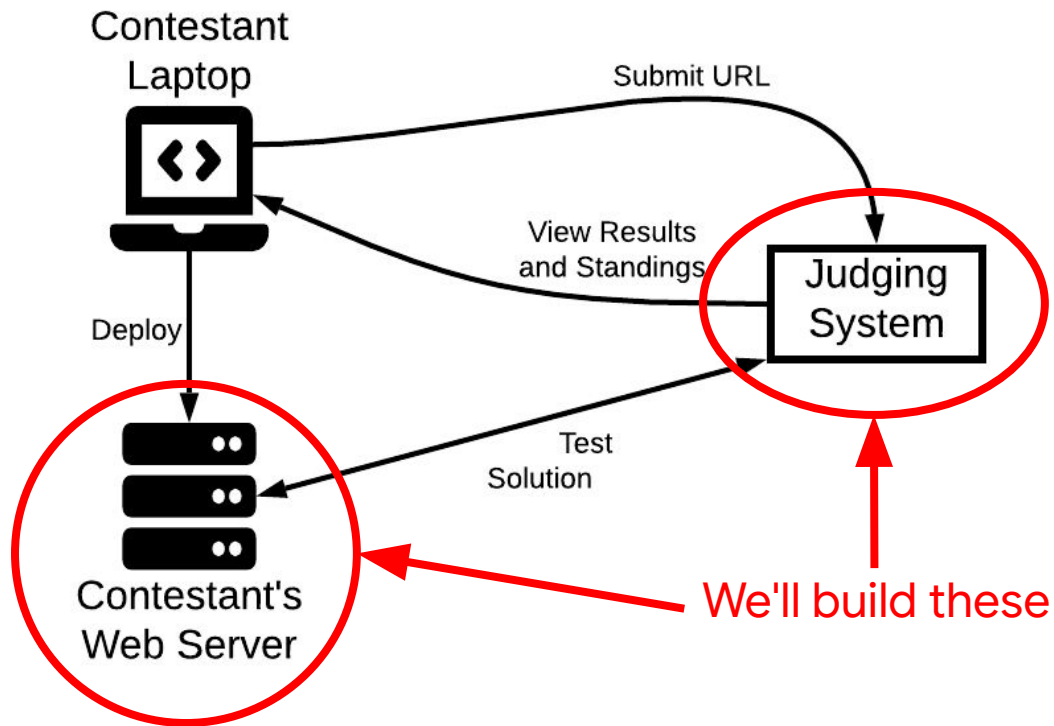
Running the submissions is a mess

- Keeping track of what was submitted, and when
 - Especially if physical media is involved
- Avoiding malicious code on the test machines
 - Or just dangerously buggy code
- Dealing with different machine configurations

Solution: don't submit programs

- Run the solutions on the **contestant's** infrastructure
 - Provide input, receive output?
 - Sounds like an HTTP(S) request
- Contestants deploy their solutions to the web
 - Provide a URL to the judges
- Judges run the code multiple times via web requests
 - (Need to slightly randomize test data so contestants don't read their logs and hard code answers)

High-level System Diagram



Is this practical?

Can we expect contestants to manage and deploy to their own web servers?

- **No**, if they have to handle system configuration and administration
- **Yes**, if they use a lightweight managed serverless platform

We will start the workshop with this part of the problem

- ***Contestant deploys solution to the web***

We will go on to the more complex judging system afterwards

We'll use Google Cloud Platform

That doesn't mean other cloud platforms couldn't be used

- They have many similar offerings
- But the steps and details would be different

Also - we work for Google. We know it best, and can provide credits to cover the cloud costs of this workshop.

Want to try this out on another platform after the workshop?

- Fork the repository and adapt it as needed
- **Let us know - we're interested!**

Workshop resources

- Your laptop with an internet connection and a modern web browser
- A Gmail account
 - Might be able to use a G Suite account, but administrators can disable Cloud Console access
 - **Set up a plain vanilla Gmail account to avoid roadblocks**
- The cloud coupon we handed out
 - **Follow the directions to apply credit to your account**
 - You have two options regarding the coupons. Wait until you understand their pros and cons before applying one

Set up your environment

Option A:

- Get \$50 credit without needing a credit card
- **Apply the coupon** at console.cloud.google.com/edu
- No chance of being charged if you don't provide a credit card

Option B:

- \$300 one year credit for new customers
- **Do not use the coupon**, go to console.cloud.google.com
- Must provide a credit card, will be charged after credit is used up

Workshop materials

These slides - serverlessworkshop.dev/slides.pdf

Source code:

github.com/GoogleCloudPlatform/serverless-game-contest

Codelabs:

Player - serverlessworkshop.dev/player

Questioner - serverlessworkshop.dev/questioner

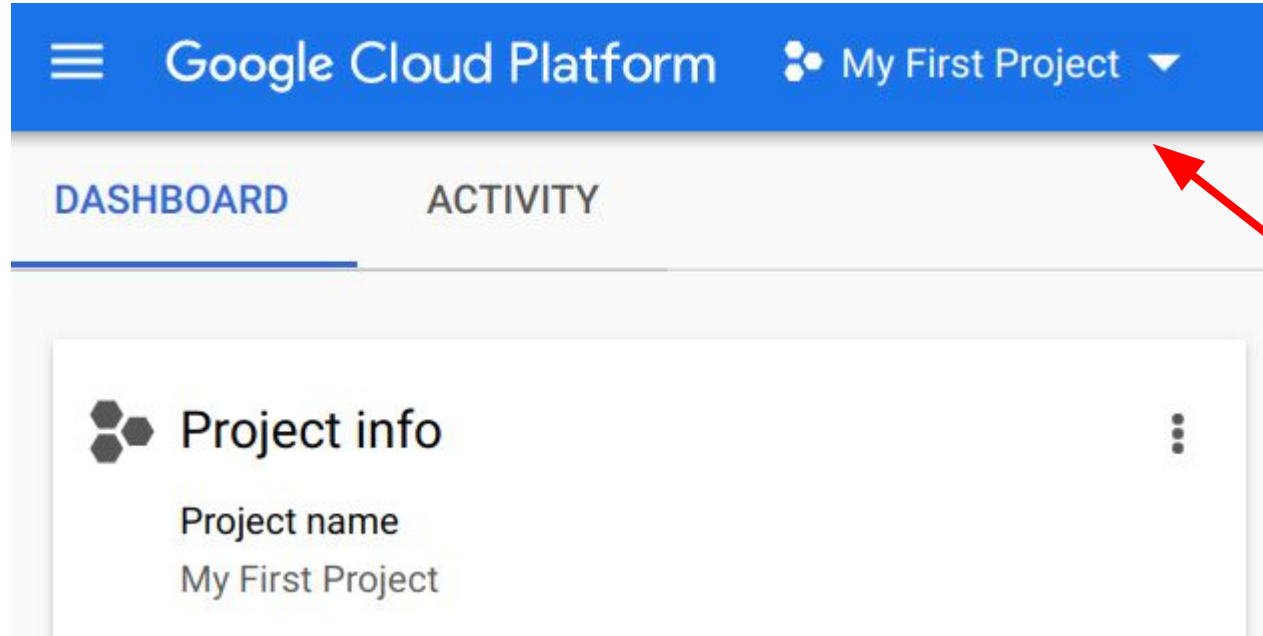
Manager - serverlessworkshop.dev/manager

serverlessworkshop.dev

GCP Projects

- All GCP resources live in **projects**
 - Resources in the same project can usually interact with each other
 - You can enable resources in different projects to interact
 - You can restrict resources in the same project from interacting
- Contestants and the judging system would, in practice, be in separate projects, owned by different entities
 - **But to keep things simple, we will create and use one project for everything in this workshop**
 - We will discuss how it could be separated, though

Creating a Project - Click the drop-down

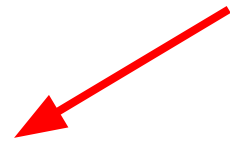


Click NEW PROJECT

Select a project




NEW PROJECT



 Search projects and folders

RECENT

ALL

Name		ID
  My First Project 	a-project-id	

Call it whatever you like

For example "serverless-workshop"

Notifications



Create Project: serverless-workshop

Just now



Create Project: My First Project

6 days ago

Click the notification when ready to open the project

The project name will be in URLs, which will show in contest results, so pick a name you're okay with others seeing!



Google Cloud

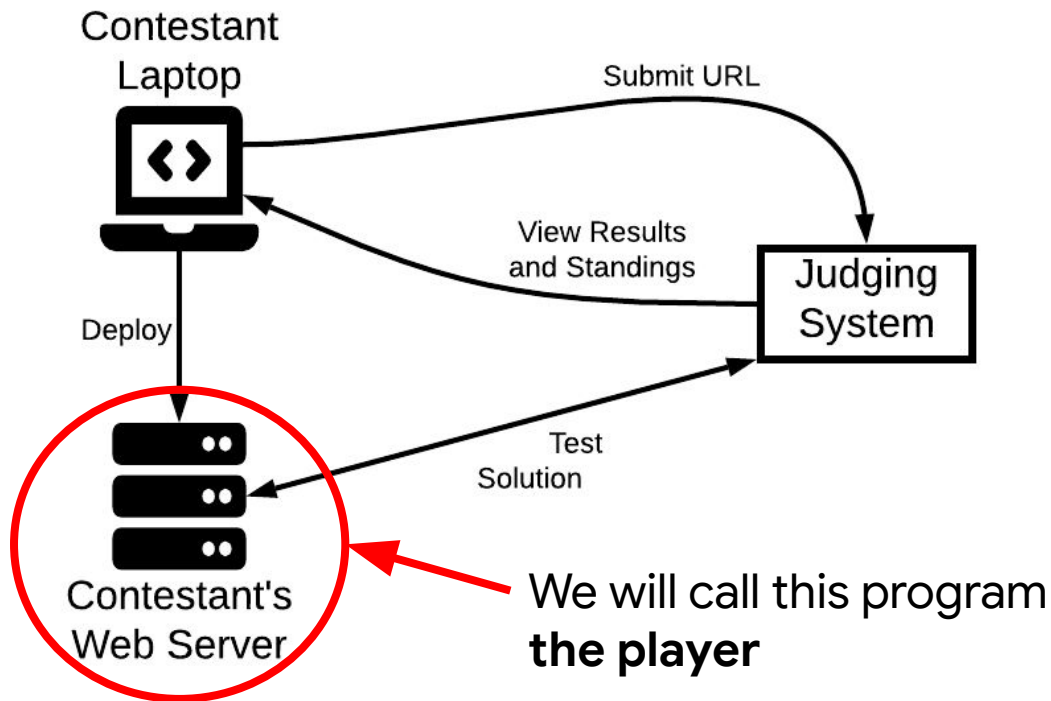
The Player

Start simple - the game player

- Contestant writes a program that accepts an HTTP request representing the game state
- Responds with a game move
- Deploys program to the internet
- Submits the program for judging by providing the URL

We will address the more complex judging system after working out the basics with this program

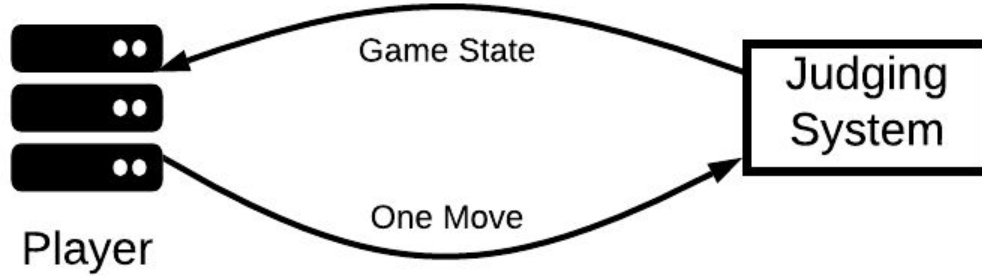
Recall the High-level System Diagram



Player platform: Google Cloud Function

- Managed serverless platform
 - Provide a program
 - Specify a triggering event (web request)
 - Platform runs the program when the event occurs
- Benefits of Cloud Function platform
 - No system administration, just write a program
 - Scales as needed automatically
 - Scales to zero when idle

How the judging system plays a game



1. Sends initial game state to player
2. Gets a move in response
3. Updates the game state, make the opponent's move if needed
4. Sends the new game state to get the next move

Coupling?

The player is nearly completely uncoupled from the judging system

- Only connection is HTTP requests over public internet

That's important, because each contestant builds a separate player

- Don't want to have them sharing resources with each other, or with the judging system

In general, **minimizing coupling between components** makes system design, deployment, and maintenance more flexible

Example game: Tic-tac-toe first move

- Initial game state is an empty board
- Represented in JSON:

```
{"marks-so-far": [], "your-mark": 'X'}
```
- Player responds with JSON representation of a move:

```
{"row": 2, "column": 2}
```

(Contest says rows and columns numbered 1, 2, 3)
- Judging system makes a move, sends updated game:

Judging system processes move

- Makes a move of its own
- Asks player for another move, given new game state:

```
{ "marks-so-far": [  
    { "mark": "X", "row": 2, "column": 2 },  
    { "mark": "O", "row": 1, "column": 1 } ],  
  "your-mark": 'X' }
```
- Player responds with another move

```
{ "row": 1, "column": 2 }
```
- Play continues until player wins, loses, fails, or crashes

Rules for our game

1. The simplest possible game: **guess a number**
2. Given minimum and maximum, and history of guesses
3. Respond with a whole number guess

We don't worry about the judging system for now (we're the contestant who has to write a player).

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"}  
  ]  
}
```

Time to Build and Deploy the Solution

Hands-on codelab at

<https://serverlessworkshop.dev/player>

Player recap

The player does only one simple thing:

- Make a move given the existing game state

The player does not keep state

- It doesn't know its previous moves, it has to be told when a new move is requested

Moves are requested by making a web request

- Cloud function platform invokes the player code when a request arrives

Want to try it out?

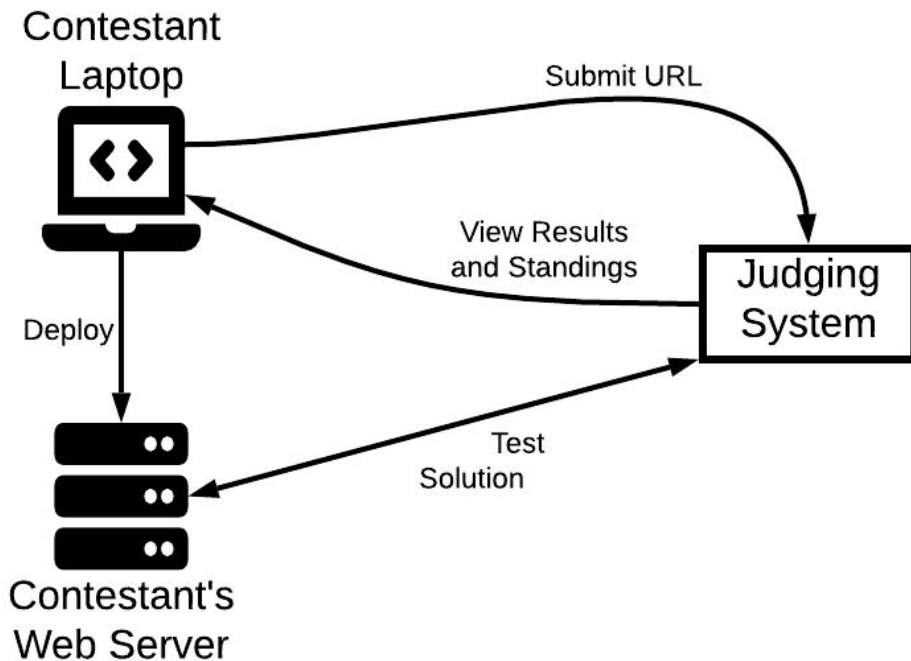
The system being built for this workshop has a live version available:

<https://serverlessworkshopdemo.appspot.com/>

You can submit the player you just wrote to be judged there

The Judging System - Part 1

Recall the High-level System Diagram



Looks like a monolith

Judging
System

- Traditional approach might be a single web server app
 - Interacts with contestants
 - Plays games against submitted solutions
 - Track scores in persistent data
- Would restrict flexibility in design and future expansion
 - Game judging components are often created by multiple parties, with different playing scenarios
 - Using a different game requires rebuilding whole system
 - Every submitted solution would have to wait for games to be played, or have concurrency programmed in

Look at the needs one at a time

- First - **Something** needs to play the game against submissions
 - Call this a **questioner**
 - Needs to know the player URL
 - Plays the game against the player on its own
 - Needs to know what to do with the result of play
- So build the questioner as an independent component
 - Provide the player URL and another URL to send the result of play for recording
 - Easy to run multiple questioners against each submission
 - **Use asynchronous request** to trigger start of play

Platform choice - Cloud Function

- Any compute service could do, but we have a program that does a single task, which is a good fit for CF
- Trigger asynchronously, though
 - Not via HTTP request
 - Cloud functions can be triggered by a variety of events
 - Trigger on message being published to a Pub/Sub topic by the rest of the judging system

Google Pub/Sub

Reliable messaging system

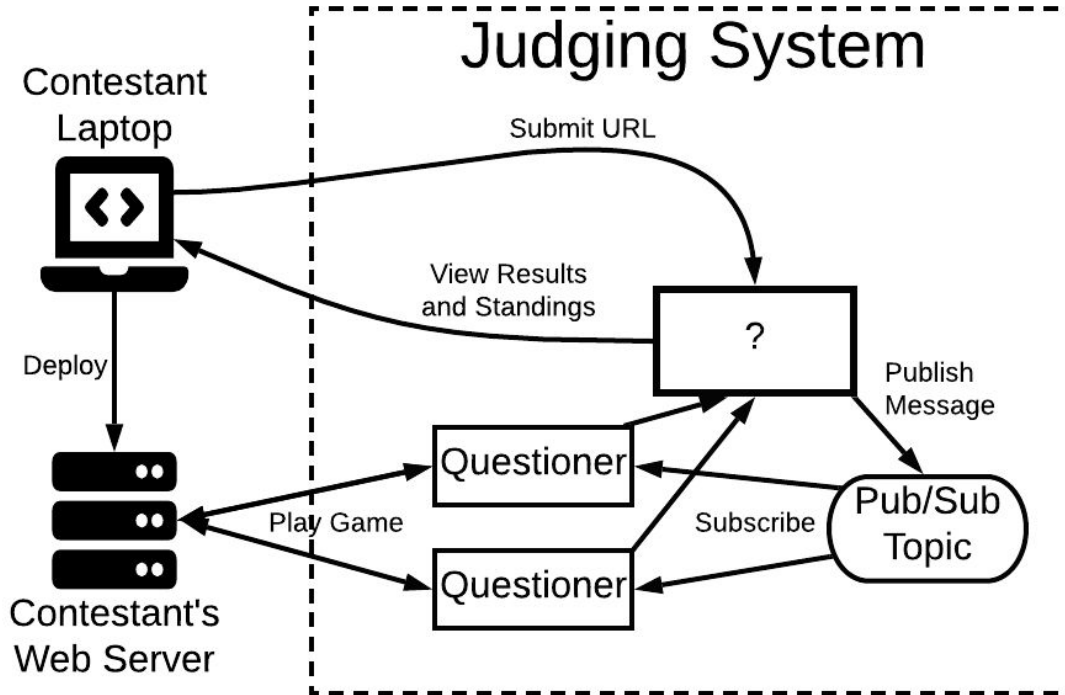
Messages belong to **topics**

- Messages are **published** to a topic
- Programs **subscribe** to a topic to get all messages
- Can be one-to-one or one-to-many

Asynchronous, reliable delivery

- Messages will be delivered to every subscriber **at least once**
- Delivery order is not guaranteed

Factor out the Questioner(s)



Message body

```
{  
  "player_url": "some-url",  
  "result_url": "another-url",  
  "contest_round": "a random ID",  
  "secret": "a shared random string"  
}
```

Coupling?

- Questioner ↔ Player?
 - HTTP requests and responses only
- Judging system ↔ Questioner?
 - Questioner must be able to subscribe to Pub/Sub topic that judging system publishes to
 - If components are in separate projects, permission to other project must be explicitly granted
 - Results get from the questioner to the judging system via HTTP POST to a provided URL

Time to Build and Deploy the Solution

Hands-on codelab at

<https://serverlessworkshop.dev/questioner>

Questioner recap

Another event-driven cloud function

- But triggered asynchronously instead of via a web request

Pub/Sub trigger lets us send one message that many questioners subscribe to

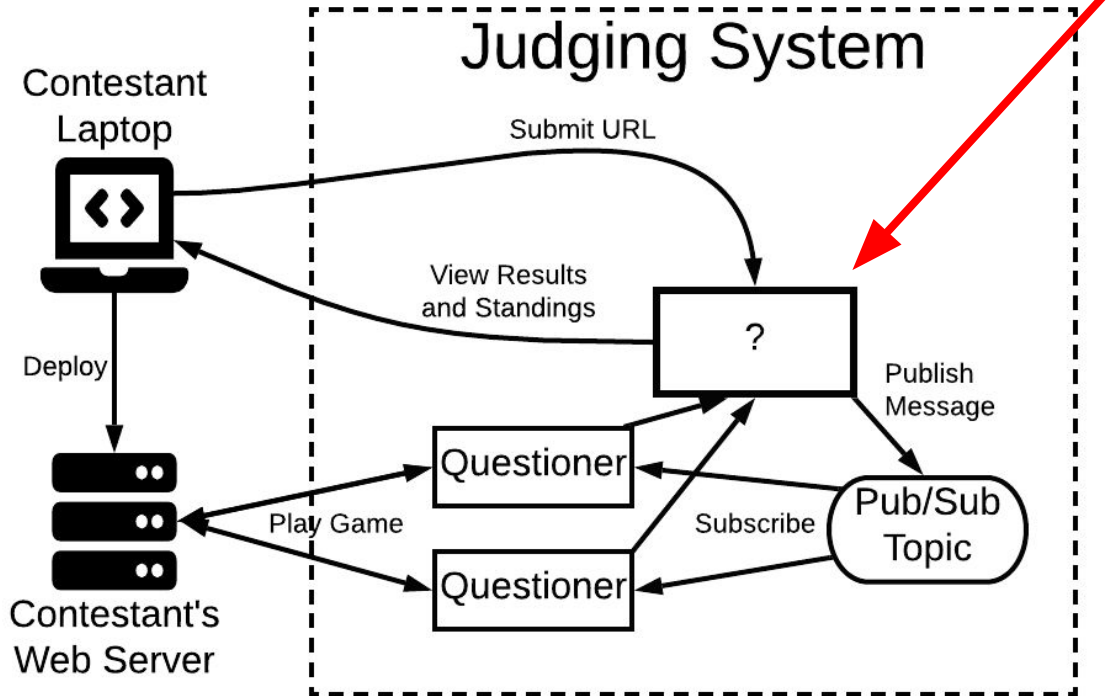
Questioners create results that need to be saved, but they aren't responsible for doing the saving

- They're told to send them to URL
- Reduces system coupling

The Judging System - Part 2

The system so far

Call the remaining judging system the **manager**



What does the manager do?

1. Displays current results on a web page
2. Lets contestants submit solutions
3. Invoke questioners by publishing messages
4. Accept results from questioners

Interact with
people

Interact with software

Can we partition it further?

Smaller pieces are easy to create and maintain

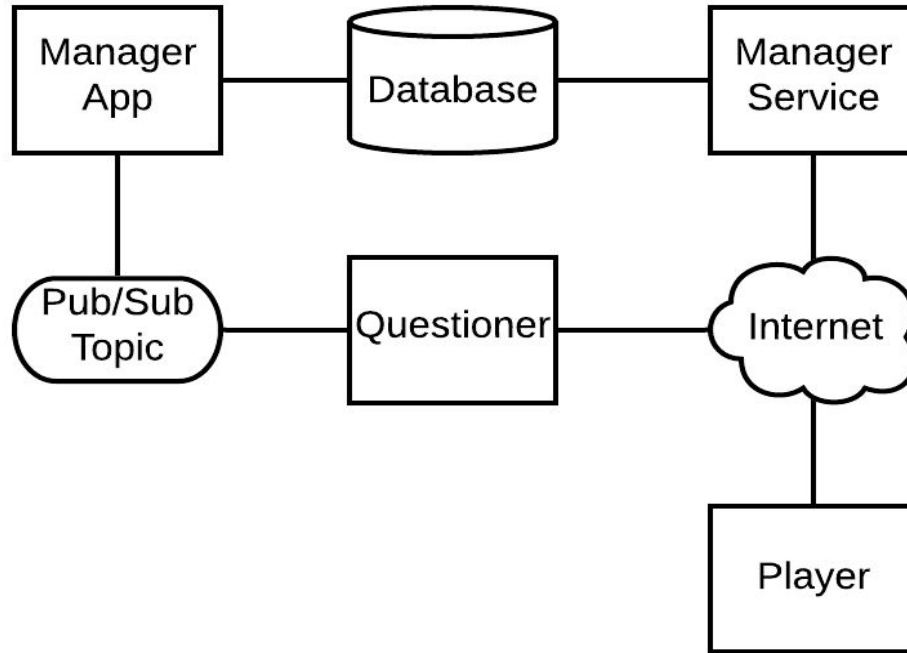
Break manager into two parts

1. Web application people interact with
2. Web service that accepts results from questioner software

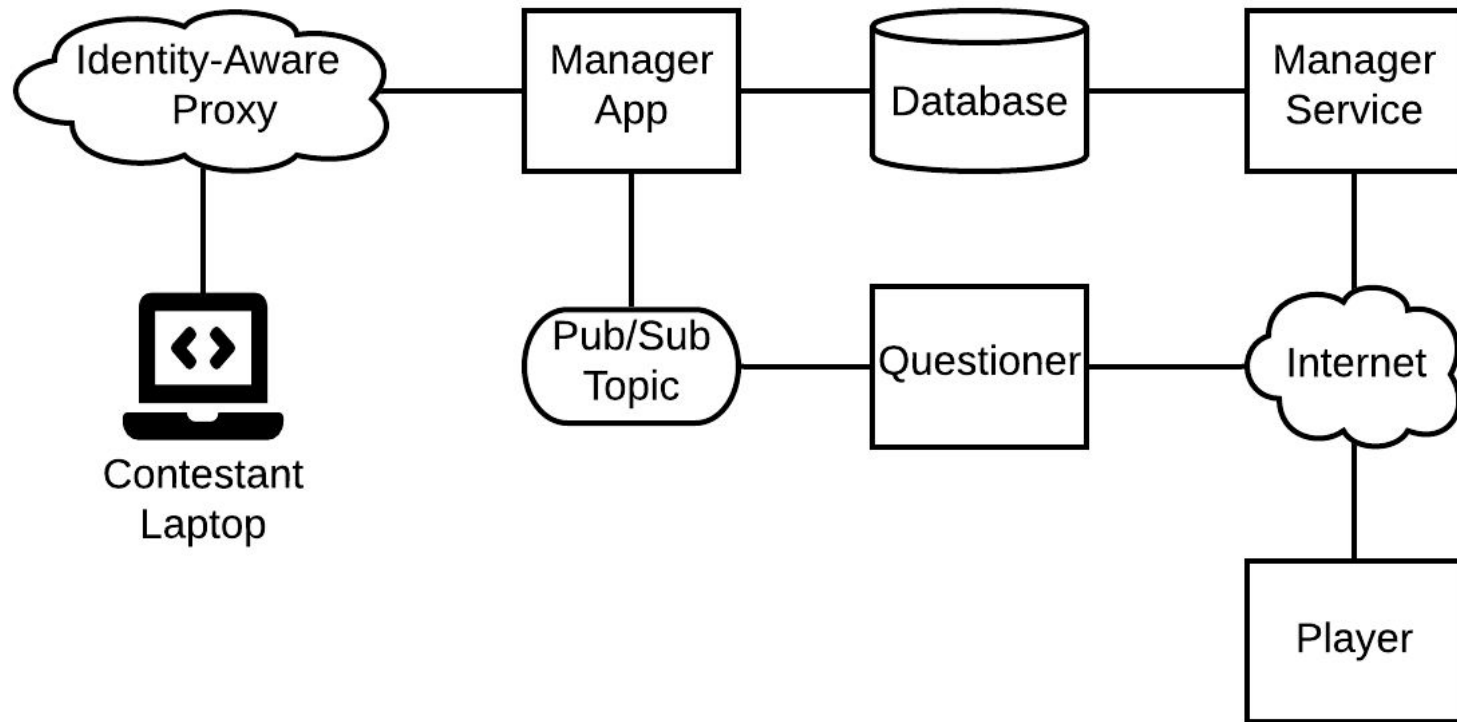
Connect via a shared database

1. Web app adds submissions to database
2. Web server adds results to submissions

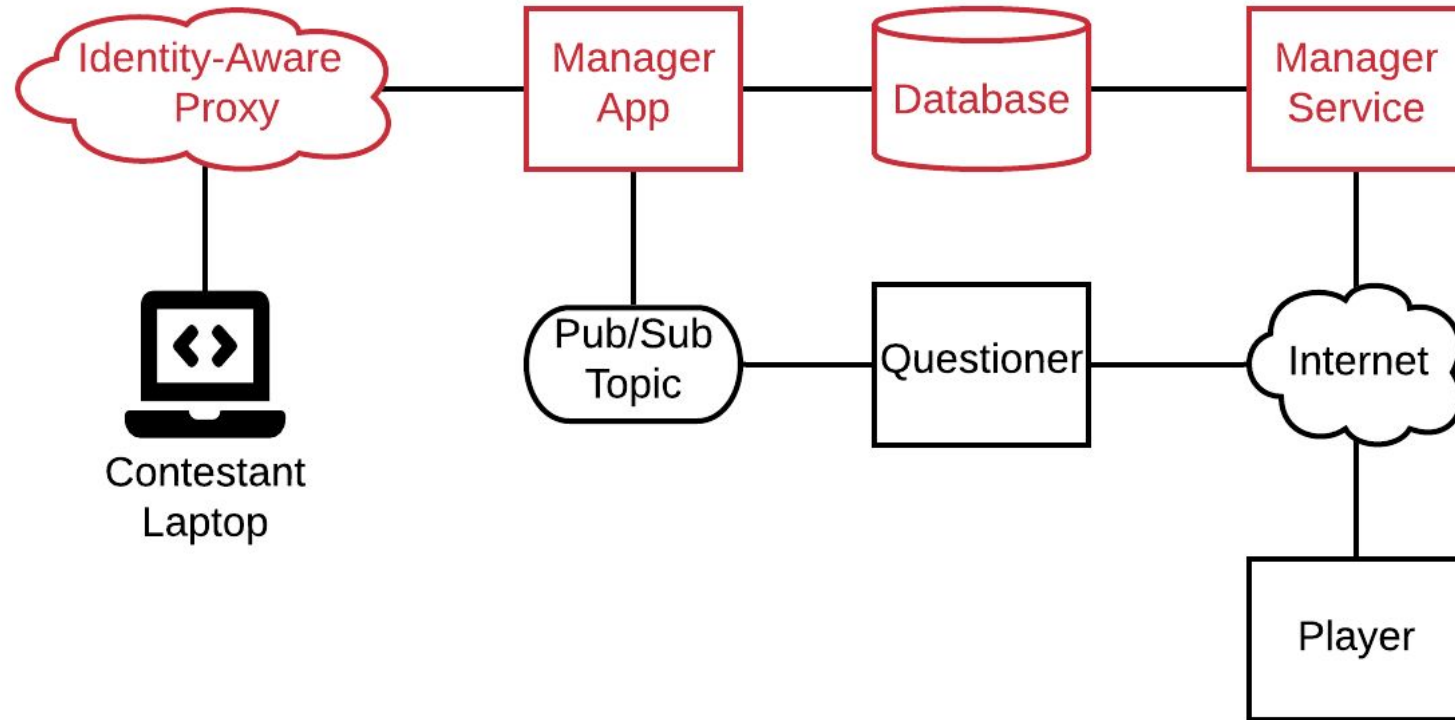
System Coupling



Add front-end user authentication



Left to deploy



Time to Build and Deploy the Solution

Hands-on codelab at

<https://serverlessworkshop.dev/manager>

Recap

Created a distributed serverless system

- Different portions owned by different entities
- Player owned by a contestant
- Manager owned by the contest runners
- Questioners delegated from the contest runners

Used several serverless tools

- Function as a service (Cloud Function), platform as a service (App Engine), reliable messaging (Pub/Sub), NoSQL database (Firestore), user authentication as a service (Identity-Aware Proxy)

Thank you!

serverlessworkshop.dev

serverlessworkshop@google.com