

Database as a Service

Database as a Service (DBaaS)

- Fully managed, NoOps, database services that automatically scale
- Many backend databases, many DBaaS
- Flavors
 - SQL
 - Cloud SQL
 - NoSQL
 - Cloud Datastore, Cloud BigTable
 - NewSQL
 - Cloud Spanner
 - Block-chain*

SQL vs. NoSQL

- SQL

- Relational structured data
- Complex querying using relations
- Schema (statically typed data)
- Strict transactional consistency
- Vertical scaling

- NoSQL

- Non-relational, unstructured data
- Simple, fast key-value lookup
- Schemaless (dynamically typed data)
- Loose eventual consistency
- Horizontal scaling

What explains the last two design patterns?

CAP Theorem (Fox/Brewer 2000)

- Can not have strong consistency in the wake of network outages with high availability
- Any networked system can have at most two of three desirable properties
 - C = consistency
 - A = availability
 - P = partition-tolerance
- Two consistency options for networked databases
 - ACID (atomicity, consistency, isolation, durability)
 - To achieve strong consistency, lose “A” availability in the face of a network partition “P”
 - Can not perform transactions until all* replicas fully on-line
 - Cloud SQL* & Cloud Spanner
 - BASE (basically available, soft state, eventual consistency)
 - To achieve high availability, lose “C” in the face of a network partition “P”
 - Cloud BigTable & Cloud Datastore

Application drives consistency model

- Bank accounts
 - Require strong consistency
- High-score updates in a game?
 - Can survive with just eventual consistency
- Different implementations of databases (and DBaaS) to support

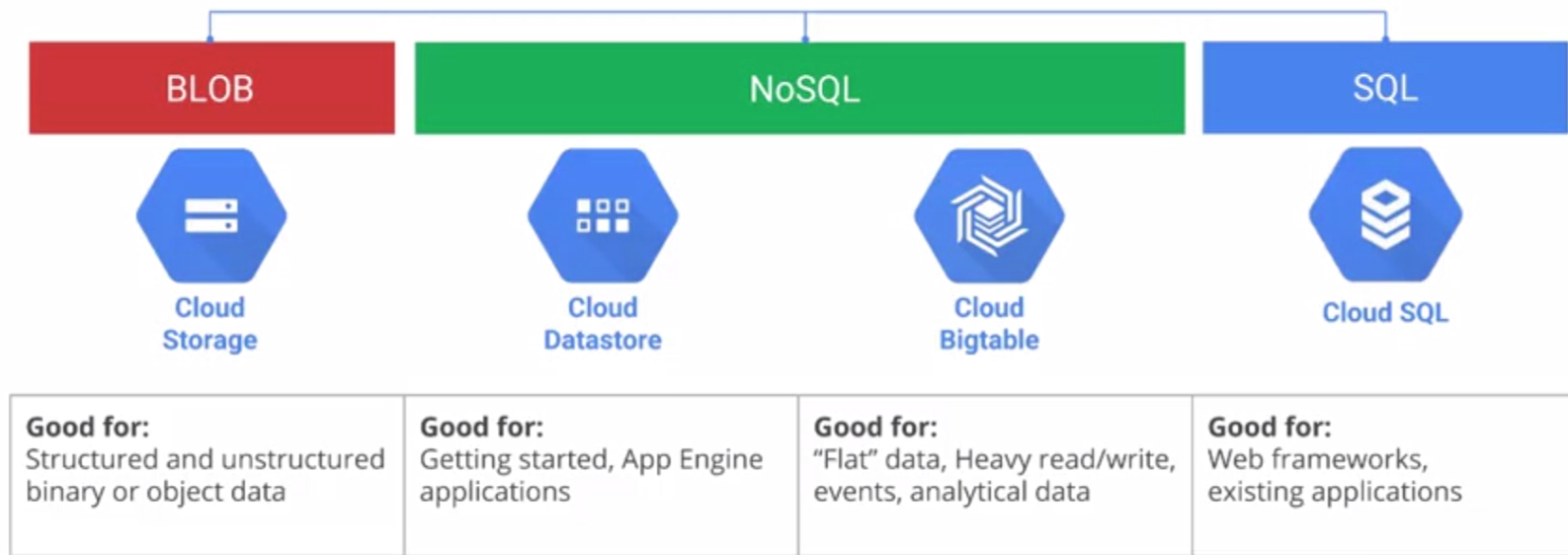
Cloud SQL

AWS RDS (Relational Database Service)
Azure SQL Database

Recall

- Fully-managed, drop-in replacement for MySQL (or Postgres) relational database
- Uses pre-configured VMs on demand
 - Vertical scaling (read and write)
 - Horizontal scaling only for reads via replicas
- Accessed via standard drivers on App Engine, SQLAlchemy, etc.

Summary



Transactions	No	Yes	No	Yes
Complex queries	No	No	No	Yes
Capacity	Petabytes+	Terabytes+	Petabytes+	Up to 500GB

Cloud Datastore (NoSQL)

AWS DynamoDB
Azure Cosmos DB

Cloud Datastore

- Distributed, managed NoSQL database optimized for reading
 - Schemaless, key-value store
 - Store entities and objects given a unique key
 - Stored object can be modified without conforming to some database schema
 - Limited querying (mostly gets and puts)
 - Like Cloud SQL: NoOps
 - Autoscaled and managed, no configuration
 - Data automatically stored across multiple zones for availability
 - Programming API from App Engine for many languages

Cloud Spanner

"NewSQL"

Cloud Spanner (2017)

- Managed, horizontally scalable, relational ACID database
- Best of SQL
 - SQL queries, JOINS
 - Schemas, strong types
 - Strong consistency
 - Indexes, strong secondary keys
- Best of NoSQL
 - Horizontal scaling

Spanner and the CAP theorem

- C (consistency) over A (availability) just like ACID
- Scale via synchronous replicas (unlike Cloud Datastore)
 - 3 copies by default
- But, when partitions happen, go into partition mode
 - Replicas use consensus mechanism to manage partitions
 - Replicas on the “majority” side of partition continue, those in minority lose availability
 - Engineer against P (partitions) via Google’s network to get 5 9s reliability
- Good for scaling OLTP (On-Line Transaction Processing) applications

<https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45855.pdf>

Cloud Spanner

- Multiple ways for accessing as with Cloud SQL and Cloud Datastore
 - REST API, Java/Go/Python/NodeJS libraries, SQL JDBC
- Cloud SQL vs Cloud Spanner
 - If data fits in single server, Cloud SQL (cheaper)
 - When vertical scaling via Cloud SQL not enough, Cloud Spanner (due to horizontal scaling ability)

Example use cases



- Require SQL with ACID at massive scale
- Initially, manually-sharded MySQL
 - Columns and tables of each database split across multiple nodes
 - Resharding a multi-year process
 - Moved to Cloud Spanner
 - F1 paper: "A Distributed SQL Database that Scales"
<https://research.google.com/pubs/pub41344.html>

Quizlet

- From sharded MySQL to Spanner
 - <https://quizlet.com/blog/quizlet-cloud-spanner>

Blockchain-as-a-Service

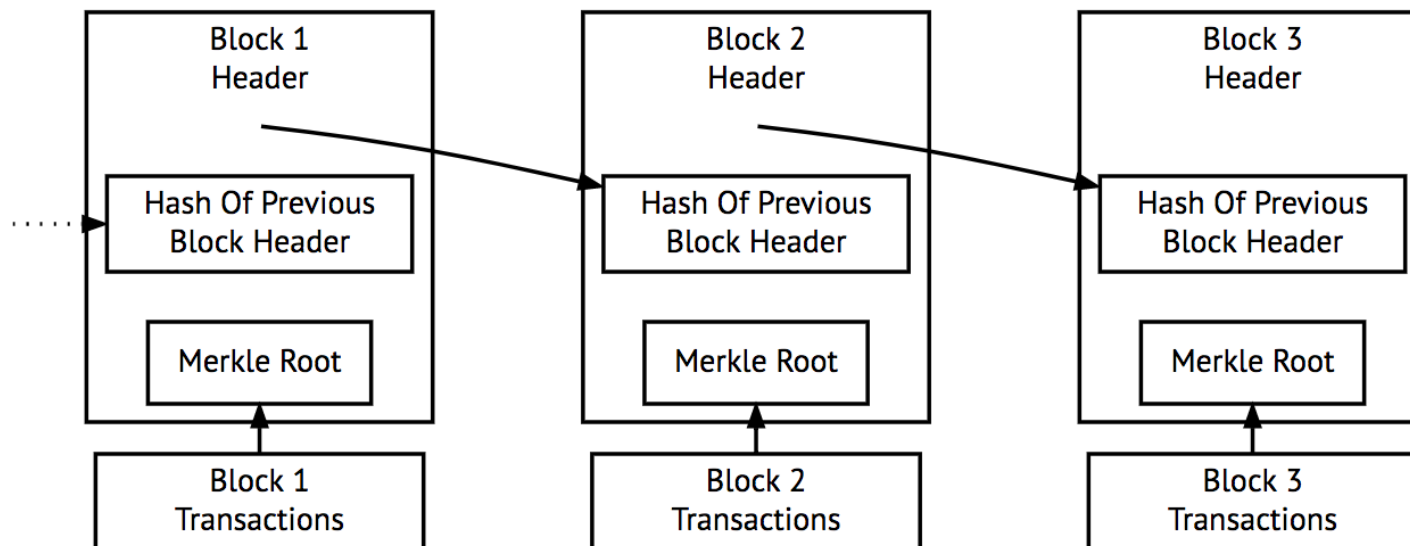
Azure Blockchain Workbench (2018)

What is it?

- Immutable ledger (transaction log)
 - Recall CRUD (create, read, update, delete)
 - Block-chain (append, read)

Essentials

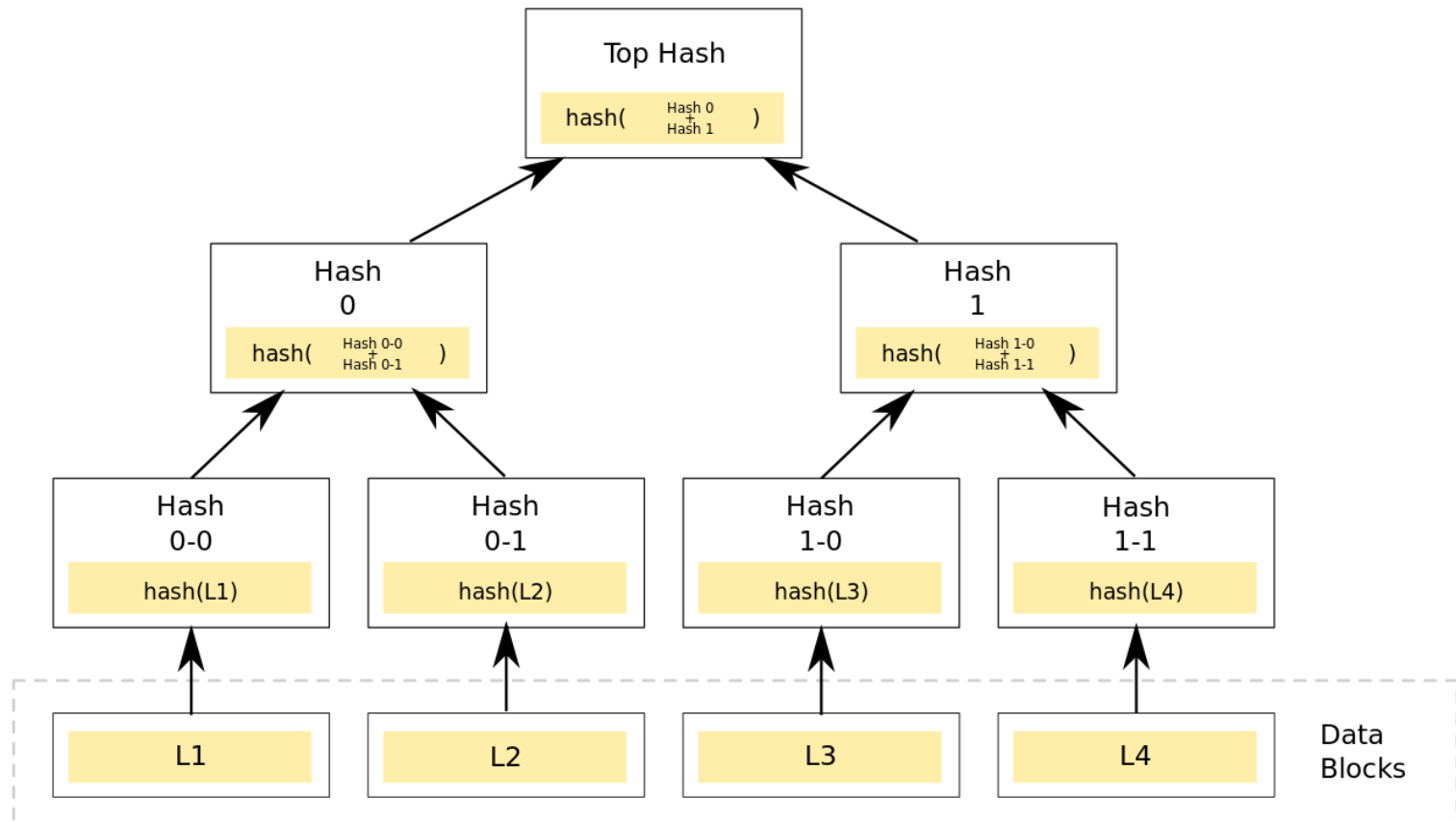
- Data stored in linked lists of blocks
 - 1 MB for original Bitcoin
- Organized as a tree, rooted at initial entry (called the base)
- Append operation protected via proof-of-work computation to prevent tampering (on public block-chains)
- New blocks stored with a cryptographic hash, derived from base, through individual lists of blocks to support immutability



Simplified Bitcoin Block Chain

Essentials

- Transactions point to records on the block-chain that trace up to the "root" (i.e. base)
 - Merkle tree of hash-chains
 - Applied to blocks to give block-chains their name



Essentials

- Entire block-chain replicated amongst a large number of independent machines for durability and immutability
 - BTC ledger @ ~150GB, 1MB every 10 min
- Consensus agreement to prevent tampering (exactly like Spanner!)
- Public-key cryptography for authenticating transactions
 - For block-chains handling financial data

Classes of applications

- Auditing for compliance and provenance
 - Leverages immutability of published data onto a common data store
 - Supply-chain tracking, medical history and records, fraud detection
 - All on the ledger instead of siloed in legacy databases
- Removal of trusted third party for non-repudiation
 - Block-chain acts as a "witness"
 - Leverages agreement amongst nodes via consensus protocol
 - Anywhere that a notary or escrow is needed, replace with a public block-chain
 - Currency transactions, ownership validation, social media posts, etc.

Types of block-chains

- Can be used to commit data and/or code
 - e.g. web transactions, smart contracts
- Can be public
 - Global crypto-currency transactions (e.g. Bitcoin)
- Can be private
 - Secure and durable audits for compliance
 - Supply-chain tracking
 - Medical history and records
 - Can do without the proof-of-work and financial incentives

Disruption in health-care...

- Unified, tamper-resistant storage of medical records
- Tracking prescription drug abuse



By Stephen O'Neal

APR 17, 2018

Data, Security, Insurance: How Blockchain Is Disrupting The Health Industry

Blockchain Will Revolutionize the Medical Industry – Here's How

May 3, 2018 | Christina Comben



HACKERNOON

SPONSORED BY
INGOT.COM

Follow



Blockchain Aims to Curb Prescription Drug Abuse

Disruption in consumer fraud...

- Good-bye knock-offs

IBM Crypto-anchors and blockchain will save up to \$600 billion per year from fraud

brian wang | March 19, 2018 [2 comments](#)

06.03.16

How Sneaker Designers Are Busting Knock-Offs With Bitcoin Tech

Shoe counterfeiters are notoriously difficult to stop, but an unlikely solution is emerging—and it's inspired by cryptocurrencies.

Disruption in asset-backed securities...

- Prove and transfer ownership of arbitrary assets
 - e.g. real-estate, fine art, equity, investment funds

Nasdaq and Citi Announce Pioneering Blockchain and Global Banking Integration

May 22, 2017, 09:48:41 AM EDT By [NASDAQ.com News](#)

Harbor raises \$28M to bring blockchain to real estate, fine art

By | April 17, 2018 | No Comments

Coming to Oregon?

≡ WILLAMETTE W

Bitcoin Mine Cheap Electr Boost?

The arrival of cryptoc
opportunity for the st

coinbase

IT Service Desk Engineer

Coinbase

Portland, Oregon [More Portland jobs >](#)

Our goal is to enable Coinbase team members to
to get their jobs done. Coinbase stores more digit

coinbase

Support AnalystPortland - Coinbase

Coinbase

Portland, OR, US [More Portland jobs >](#)

Motivated by Coinbase's mission and creating a s
customer base. As a Customer Support Analyst, y
neuvoo.com

coinbase

Office Manager

Coinbase

Portland, Oregon [More Portland jobs >](#)

We have a healthy foundation and we're looking
us continue to build the most effective People Op

coinbase

Manager Customer Support

Coinbase

Portland, Oregon [More Portland jobs >](#)

Motivated by Coinbase's mission and creating a s
customer base. Having opened offices in New Yor

gon for
Them a

i strange

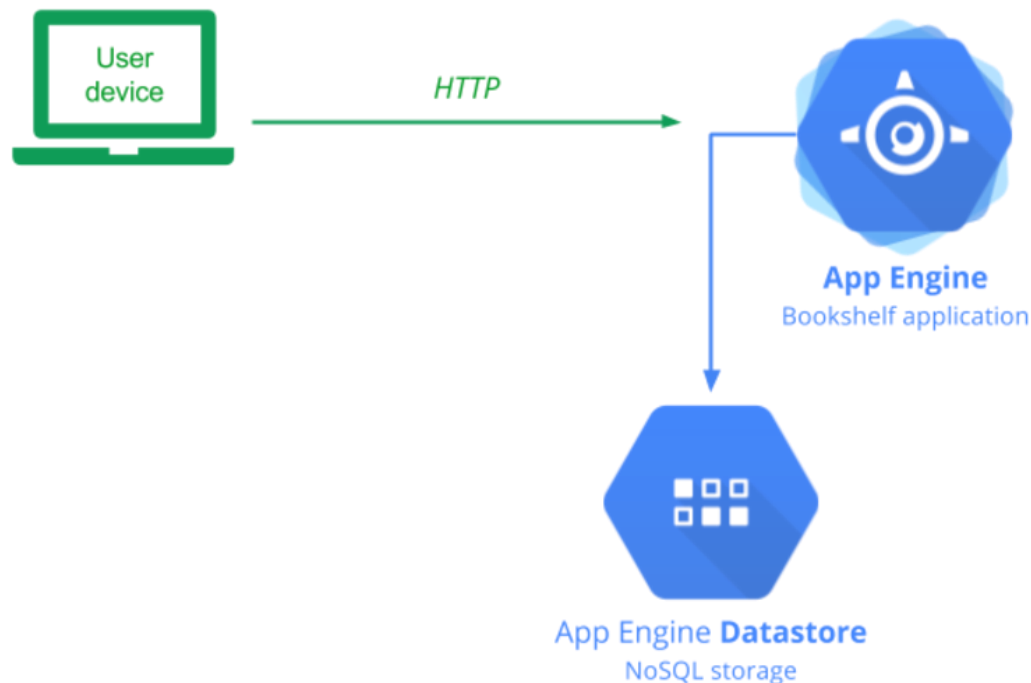
Services

- Hyperledger
 - <https://www.hyperledger.org/>
- Azure
 - <https://azure.microsoft.com/en-us/solutions/blockchain/>
- IBM
 - <https://www.ibm.com/blockchain/>
- AWS
 - <https://aws.amazon.com/partners/blockchain/>

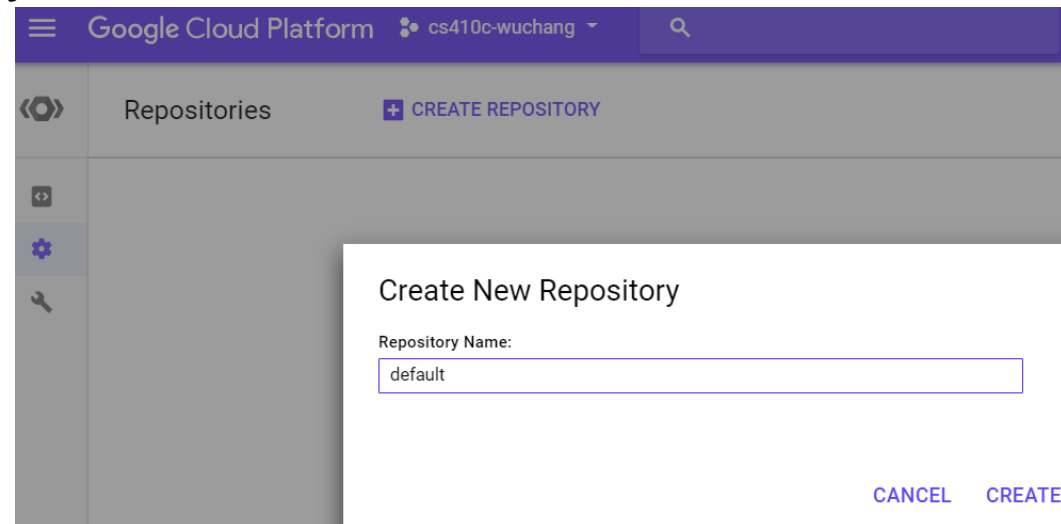
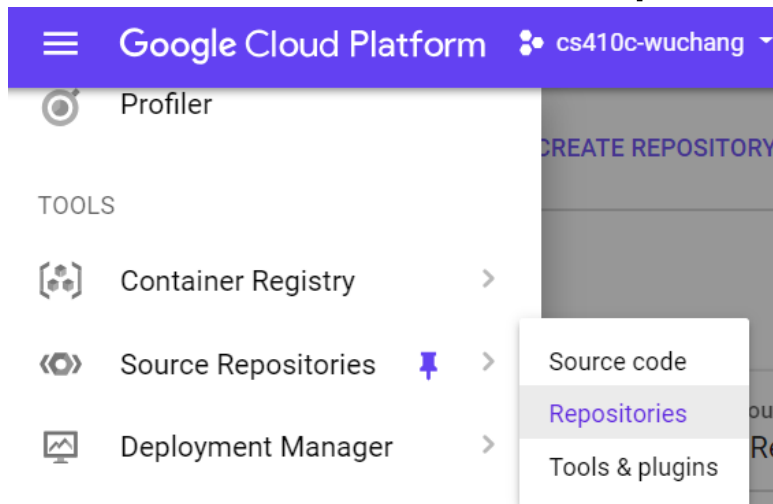
Labs

Cloud Datastore Lab #1

- Bookshelf Python/Flask app running on App Engine via managed, DBaaS NoSQL backend (Cloud Datastore) (45 min)



- Run within your class project (not cp100)
- On, navigation pane go straight to “Source Repositories => Repositories”
 - Create a new repository named "default"



- Note the options for populating your repository
 - We will be doing this via command-line

- In Cloud Shell, populate the repository

```
mkdir cp100
```

```
cd cp100
```

```
# gcloud equivalent to git clone <name_of_repo>
```

```
#     for GCP source repositories
```

```
gcloud source repos clone default
```

```
cd default
```

```
# pull the bookshelf code from Github
```

```
git pull https://github.com/GoogleCloudPlatformTraining/cp100-bookshelf
```

```
# then upload it back to the GCP source repository you just created
```

```
git push origin master
```

- Then go back to Web UI and view the files in "Source Repositories"

- Bookshelf code
 - Has versions for multiple cloud architectures
 - `app-engine`
 - PaaS (App Engine)
 - `cloud-storage`
 - PaaS with static content (App Engine w/ Cloud Storage)
 - `compute-engine`
 - IaaS (Compute Engine)
 - `container-engine`
 - Containers (Container Engine)
 - Done via simple MVC framework to separate model (database code) so that it is easily pluggable

- Within app-engine
 - `app.yaml` configures app and routes requests to it
 - All routes go to `main.app`

```
19 runtime: python27
20 api_version: 1
21 threadsafe: true
22
23 handlers:
24 - url: /*
25   script: main.app
```

- Database implementation

- Set in `config.py` (not needed in Homework #6)

```
# There are two different ways to store the data in the application.
# You can choose 'datastore', or 'cloudsql'. Be sure to
# configure the respective settings for the one you choose below.
# You do not have to configure the other data backend. If unsure, choose
# 'datastore' as it does not require any additional configuration.
DATA_BACKEND = 'datastore'
```

- `main.py`
 - Code mostly in `bookshelf` class
 - Imports `config.py` for model configuration
 - Note that `bookshelf` is imported as a directory
 - By default, Python will look for `__init__.py` for its implementation

```
15 import bookshelf
16 import config
17
18
19 app = bookshelf.create_app(config)
20
21
22 # This is only used when running locally. When running live, gunicorn runs
23 # the application.
24 if __name__ == '__main__':
25     app.run(host='127.0.0.1', port=8080, debug=True)
26
```

- bookshelf/___init___**.py**
 - **Initializes app and configures model based on config**
 - (e.g. Cloud SQL vs Cloud Datastore)

```
61 def get_model():
62     model_backend = current_app.config['DATA_BACKEND']
63     if model_backend == 'cloudsql':
64         from . import model_cloudsql
65         model = model_cloudsql
66     elif model_backend == 'datastore':
67         from . import model_datastore
68         model = model_datastore
69     else:
70         raise ValueError(
71             "No appropriate databackend configured. "
72             "Please specify datastore, or cloudsql")
73
74     return model
```

```

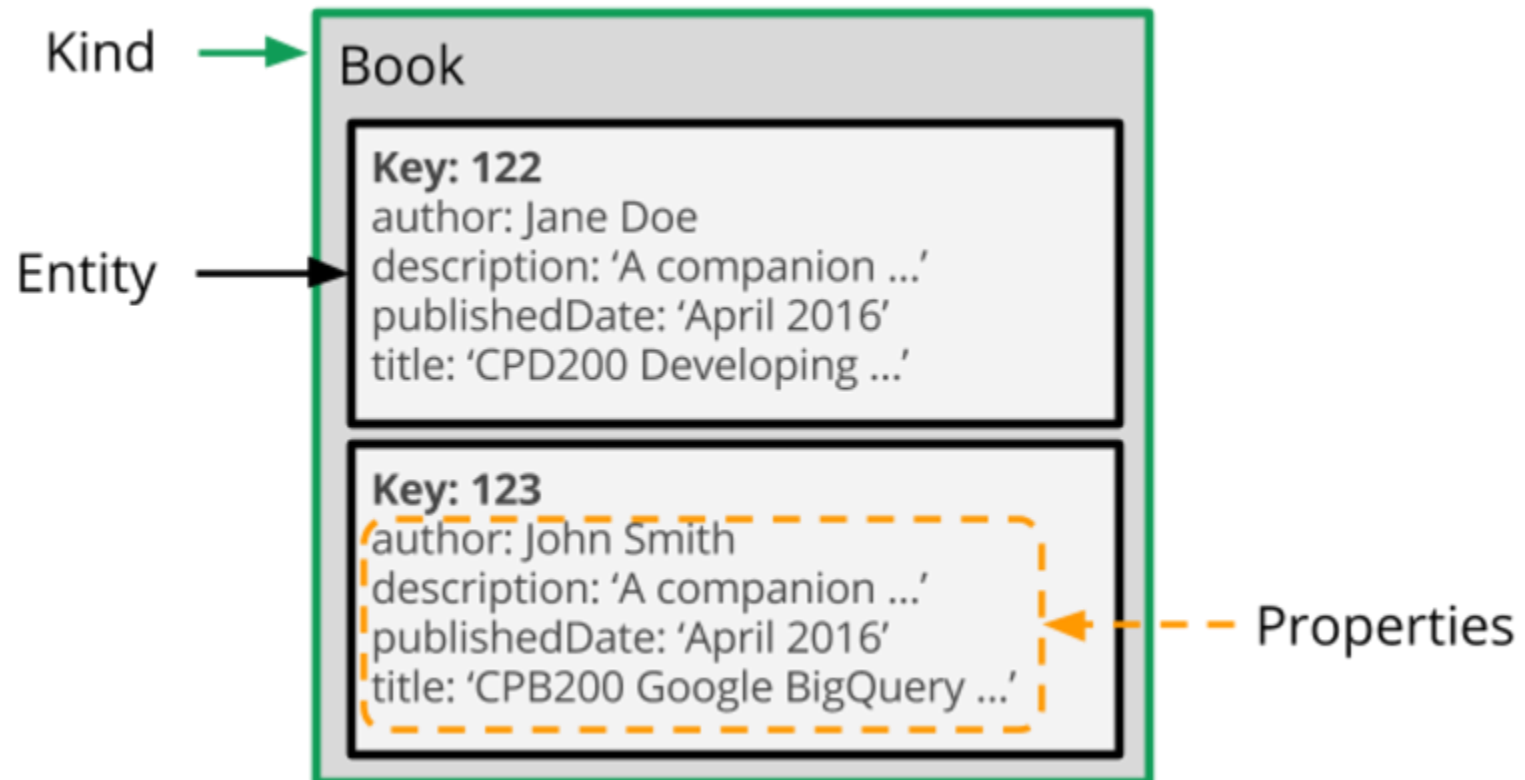
22 # [START list]
23 @crud.route("/")
24 def list():
25     token = request.args.get('page_token', None)
26     books, next_page_token = get_model().list(cursor=token)
27
28     return render_template(
29         "list.html",
30         books=books,
31         next_page_token=next_page_token)
32 # [END list]
33
34
35 @crud.route('/<id>')
36 def view(id):
37     book = get_model().read(id)
38     return render_template("view.html", book=book)
39
40
41 # [START add]
42 @crud.route('/add', methods=['GET', 'POST'])
43 def add():
44     if request.method == 'POST':
45         data = request.form.to_dict(flat=True)
46
47         book = get_model().create(data)
48
49         return redirect(url_for('.view', id=book['id']))
50
51     return render_template("form.html", action="Add", book={})
52 # [END add]

```

- `crud.py` routes for
 - `list()` -ing all books
 - `read()` -ing a single book by ID
 - `create()` -ing a book
 - `delete()` -ing a book
 - `edit()` -ing a book
 - Note use of `get_model()` throughout to abstract out which backend database is used

- Each model implements same 5 methods
- Database implementation in `model_datastore.py`
 - Implementation for managed NoSQL (Cloud Datastore)
 - Recall key-value storage abstraction
 - Key is a unique integer
 - Google's `ndb` Python client library for interfacing with Cloud Datastore
 - Note the restricted interface to backend datastore
 - `get()`
 - `put()`
 - `delete()`

- Cloud Datastore model
 - Kind = similar to table in SQL, categorizes entities for queries
 - Entities = similar to a row in SQL, but not all entities of a Kind have the same properties. Has a unique key.
 - Properties = similar to columns in SQL



```
from google.appengine.datastore.datastore_query import Cursor
from google.appengine.ext import ndb
```

```
# Creates a Book "Kind" from base Datastore model class
```

```
class Book(ndb.Model):
    author = ndb.StringProperty()
    description = ndb.StringProperty(indexed=False)
    publishedDate = ndb.StringProperty()
    title = ndb.StringProperty()
```

```

from google.appengine.datastore.datastore_query import Cursor
from google.appengine.ext import ndb

# Creates a derived class from base Datastore model class
class Book(ndb.Model):
    author = ndb.StringProperty()
    description = ndb.StringProperty(indexed=False)
    publishedDate = ndb.StringProperty()
    title = ndb.StringProperty()

# Lookup key based on Kind 'Book' and id (given as a string)
# get() a Book Entity by ID, conver to a Python dictionary
def read(id):
    book_key = ndb.Key('Book', int(id))
    results = book_key.get()
    return from_datastore(results)

# Translates datastore Entity to a Python dict for application.
#   Datastore format: [Entity{key: (kind, id), prop: val, ...}]
#   Returns: {id: id, prop: val, ...}
def from_datastore(entity):
    ...
    book = {}
    book['id'] = entity.key.id()
    book['author'] = entity.author
    ...
    return book

```



```
# If ID given, get() Book entity otherwise create new Book entity
# then set fields based on data, before put()
```

```
def update(data, id=None):
    if id:
        key = ndb.Key('Book', int(id))
        book = key.get()
    else:
        book = Book()
    book.author = data['author']
    book.description = data['description']
    book.publishedDate = data['publishedDate']
    book.title = data['title']
    book.put()
    return from_datastore(book)

def delete(id):
    key = ndb.Key('Book', int(id))
    key.delete()
```

- **Alternate database implementation in `model_cloudsql.py`**
 - Implementation for managed SQL (Cloud SQL)
 - SQLAlchemy (Python support for writing to SQL backends)

```
from flask.ext.sqlalchemy import SQLAlchemy
db = SQLAlchemy() # [START read]
```

```
def read(id):
    result = Book.query.get(id)
    if not result:
        return None
    return from_sql(result)
# [END read]
def delete(id):
    Book.query.filter_by(id=id).delete()
    db.session.commit()
# [END delete]
```

- Bookshelf code

- Python modules specified in `requirements.txt`

```
Flask==0.11.1  
gunicorn==19.6.0
```

- Install packages in `requirements.txt` in `lib` directory

```
cd ~/cp100/default/app-engine  
pip install -r requirements.txt -t lib
```

- `appengine_config.py` then loads `lib` directory packages when app deployed

```
11 # Third-party libraries are stored in "lib", vendoring will make  
12 # sure that they are importable by the application.  
13 vendor.add('lib')
```

- Then, deploy the app

```
gcloud app deploy
```

- Visit the web application after deployed
 - Add the book as described in the walkthrough
- Turn in
 - Submit a book to your app
 - Then, go to Storage => Datastore => Entities to show the book added
 - Add a book via this interface and return to the web app
 - Show both books
- Remove the app from App Engine (see prior lab)

Cloud Datastore Lab #1

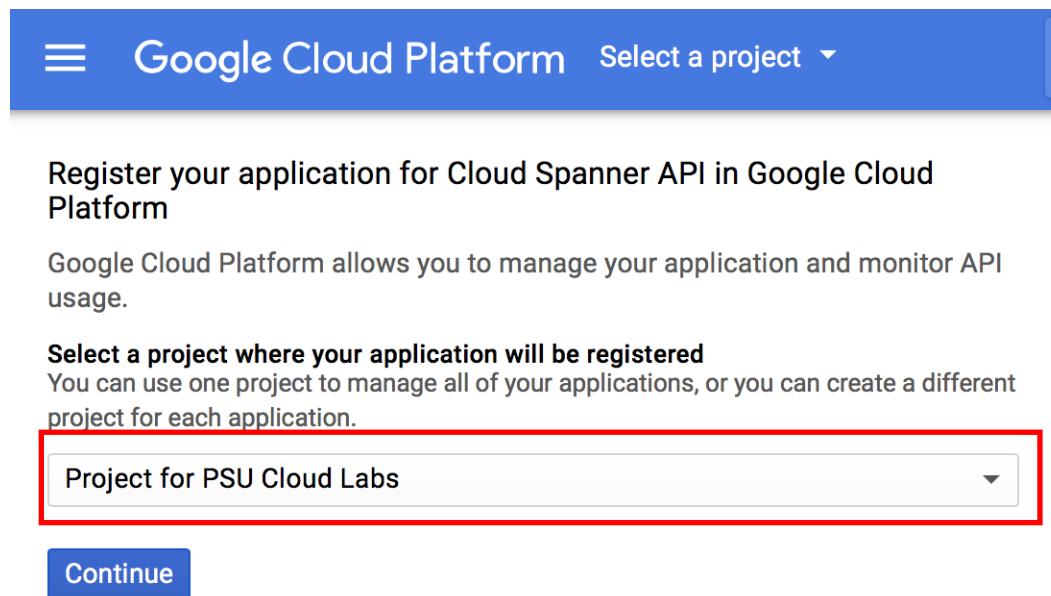
- PaaS+DBaaS
 - Bookshelf Python/Flask app running on App Engine via managed, DBaaS NoSQL backend (Cloud Datastore) (45 min)
- Link to lab
 - <https://codelabs.developers.google.com/codelabs/cp100-app-engine>

Homework #6 (510 only)

- Adapt your app from Homework #3 to work on App Engine using App Engine's Datastore
 - Leave it up for the instructor to test
- Commit your code to Bitbucket under directory `hw6`
 - Place all code and configuration files in repo
 - Submit a file called `url.txt` repository containing the URL that points to your running instance

Spanner Lab #1

- Getting Started with Cloud Spanner in Python
 - Uses multiple methods for accessing
- Enable API
 - Use us-west1 region



☰ Google Cloud Platform Select a project ▼

Register your application for Cloud Spanner API in Google Cloud Platform

Google Cloud Platform allows you to manage your application and monitor API usage.

Select a project where your application will be registered
You can use one project to manage all of your applications, or you can create a different project for each application.

Project for PSU Cloud Labs ▼

Continue

- In Cloud Shell, set up authentication and authorization (if needed)

```
gcloud config set project <Project_ID>
gcloud auth application-default login
```

If you decide to proceed anyway, your user credentials may be visible to others with access to this virtual machine. Are you sure you want to authenticate with your personal account?

Do you want to continue (Y/n)? Y

Go to the following link in your browser:

<https://accounts.google.com/o/oauth2/auth>

Enter verification code:

Credentials saved to file: [/tmp/tmp. /application_default_credentials.json]

These credentials will be used by any library that requests Application Default Credentials.

To generate an access token for other uses, run:

```
gcloud auth application-default print-access-token
```

```
@dbaaslab3:~$
```


Setup

- Clone the sample app repository (not necessary)

```
git clone https://github.com/GoogleCloudPlatform/python-docs-samples.git
```

- Set up a **local** Python virtual environment and install Spanner dependencies

```
cd python-docs-samples/spanner/cloud-client
virtualenv env
source env/bin/activate
pip install -r requirements.txt
```

- Create a 1-node Cloud Spanner instance in us-west1

```
gcloud spanner instances create test-instance \
  --config=regional-us-west1 \
  --description="Test Instance" --nodes=1
```

```
Creating instance...done.
```

Python code for creating database

(SQL) `python snippets.py test-instance --database-id example-db
create_database`

spanner/cloud-client/snippets.py

[VIEW ON GITHUB](#)

```
def create_database(instance_id, database_id):
    """Creates a database and tables for sample data."""
    spanner_client = spanner.Client()
    instance = spanner_client.instance(instance_id)

    database = instance.database(database_id, ddl_statements=[
        """CREATE TABLE Singers (
            SingerId      INT64 NOT NULL,
            FirstName     STRING(1024),
            LastName      STRING(1024),
            SingerInfo     BYTES(MAX)
        ) PRIMARY KEY (SingerId)""",
        """CREATE TABLE Albums (
            SingerId      INT64 NOT NULL,
            AlbumId       INT64 NOT NULL,
            AlbumTitle     STRING(MAX)
        ) PRIMARY KEY (SingerId, AlbumId),
        INTERLEAVE IN PARENT Singers ON DELETE CASCADE"""
    ])

    operation = database.create()

    print('Waiting for operation to complete...')
    operation.result()

    print('Created database {} on instance {}'.format(
        database_id, instance_id))
```

Spanner database client (SQL)

- Client class used to interact with Spanner database

```
cd spanner/cloud-client/  
python quickstart.py
```

spanner/cloud-client/quickstart.py

[VIEW ON GITHUB](#)

```
# Imports the Google Cloud Client Library.  
from google.cloud import spanner  
  
# Instantiate a client.  
spanner_client = spanner.Client()  
  
# Your Cloud Spanner instance ID.  
instance_id = 'my-instance-id'  
  
# Get a Cloud Spanner instance by ID.  
instance = spanner_client.instance(instance_id)  
  
# Your Cloud Spanner database ID.  
database_id = 'my-database-id'  
  
# Get a Cloud Spanner database by ID.  
database = instance.database(database_id)  
  
# Execute a simple SQL statement.  
with database.snapshot() as snapshot:  
    results = snapshot.execute_sql('SELECT 1')  
  
    for row in results:  
        print(row)
```

Python client for inserting data

- Via a **Batch** object
 - Container for mutation operations (create/insert, update, delete) to be applied atomically to a set of rows/tables

```
spanner_client = spanner.Client()
instance = spanner_client.instance(instance_id)
database = instance.database(database_id)

with database.batch() as batch:
    batch.insert(
        table='Singers',
        columns=('SingerId', 'FirstName', 'LastName',),
        values=[
            (1, u'Marc', u'Richards'),
            (2, u'Catalina', u'Smith'),
```

- Run snippets.py with insert_data as argument

```
python snippets.py test-instance --database-id
example-db insert_data
```

Inserted data.

CLI for querying data

- Via command line, execute arbitrary SQL on Spanner instance to read values columns from the Albums table

```
gcloud spanner databases execute-sql example-db  
  --instance=test-instance  
  --sql='SELECT SingerId, AlbumId, AlbumTitle FROM Albums'
```

- Show the results

```
SingerId AlbumId AlbumTitle  
1         1       Total Junk  
1         2       Go, Go, Go  
2         1       Green  
...
```

Python client for querying data (SQL)

- `query_data` to get album information via SQL

```
spanner_client = spanner.Client()
instance = spanner_client.instance(instance_id)
database = instance.database(database_id)

with database.snapshot() as snapshot:
    results = snapshot.execute_sql(
        'SELECT SingerId, AlbumId, AlbumTitle FROM Albums')

    for row in results:
        print(u'SingerId: {}, AlbumId: {}, AlbumTitle: {}'.format(*row))
```

- Run `snippets.py` using the `query_data` argument

```
python snippets.py test-instance --database-id
example-db query_data
```

- Show results

```
SingerId: 2, AlbumId: 2, AlbumTitle: Forever Hold Your Peace
SingerId: 1, AlbumId: 2, AlbumTitle: Go, Go, Go
SingerId: 2, AlbumId: 1, AlbumTitle: Green
...
```

Reading data via Python

- `read_data` to get album information via Spanner API

```
with database.snapshot() as snapshot:
    keyset = spanner.KeySet(all_=True)
    results = snapshot.read(
        table='Albums',
        columns=('SingerId', 'AlbumId', 'AlbumTitle',),
        keyset=keyset,)

    for row in results:
        print(u'SingerId: {}, AlbumId: {}, AlbumTitle: {}'.format(*row))
```

- Run script using the `read_data` argument

```
python snippets.py test-instance --database-id
example-db read_data
```

- Show results

```
SingerId: 1, AlbumId: 1, AlbumTitle: Total Junk
SingerId: 1, AlbumId: 2, AlbumTitle: Go, Go, Go
SingerId: 2, AlbumId: 1, AlbumTitle: Green
...
```

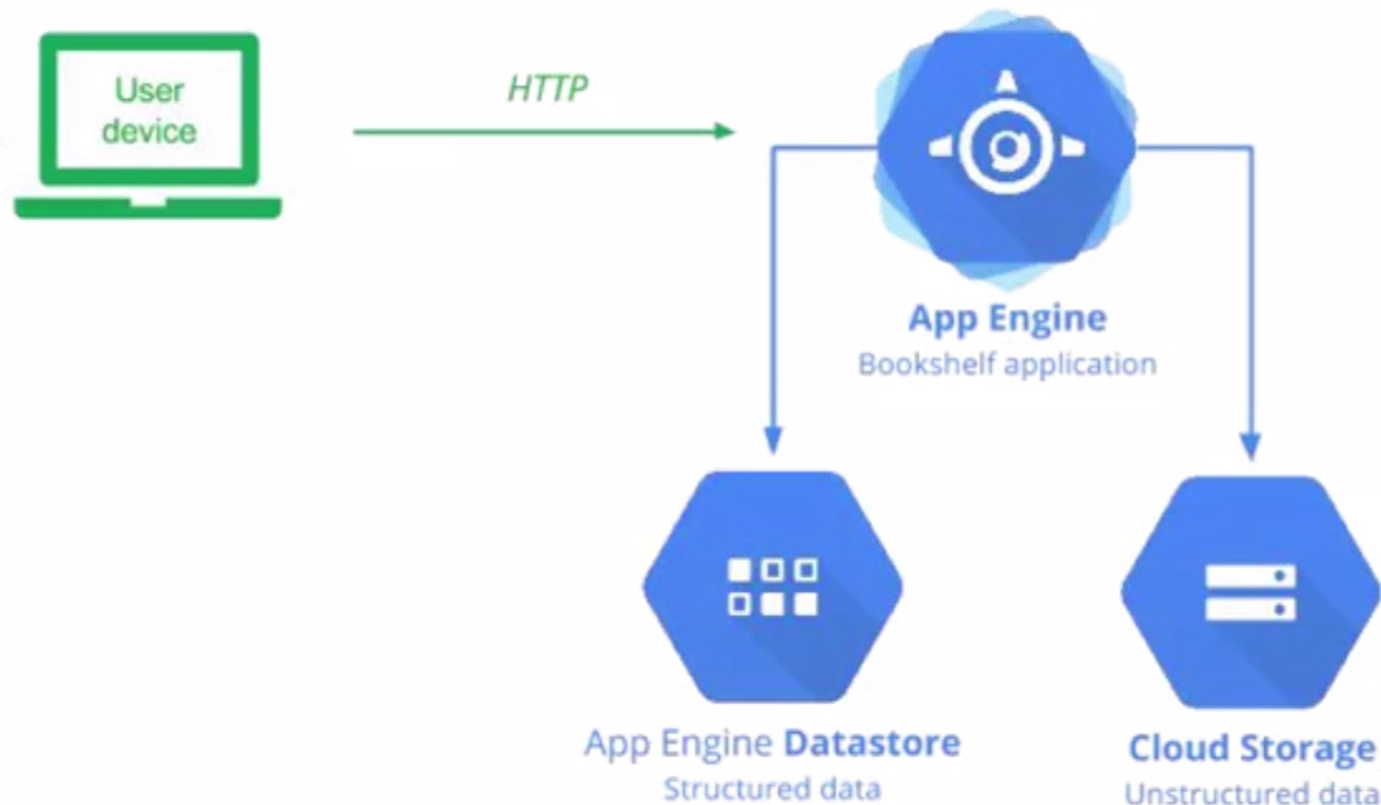
Cloud Spanner Lab #1

- Only do the first two bullets of the walk-through
 - <https://cloud.google.com/spanner/docs/getting-started/python/>
 - Note, you may do the entire lab on Cloud Shell
 - (i.e stop at "Update the database schema" section)
- Remember to delete everything when done
 - \$\$\$

Extra

PaaS+DBaaS+Cloud Storage Lab #1

- Add integration with Google Cloud Storage (35 min)
 - <https://codelabs.developers.google.com/codelabs/cp100-cloud-storage/>



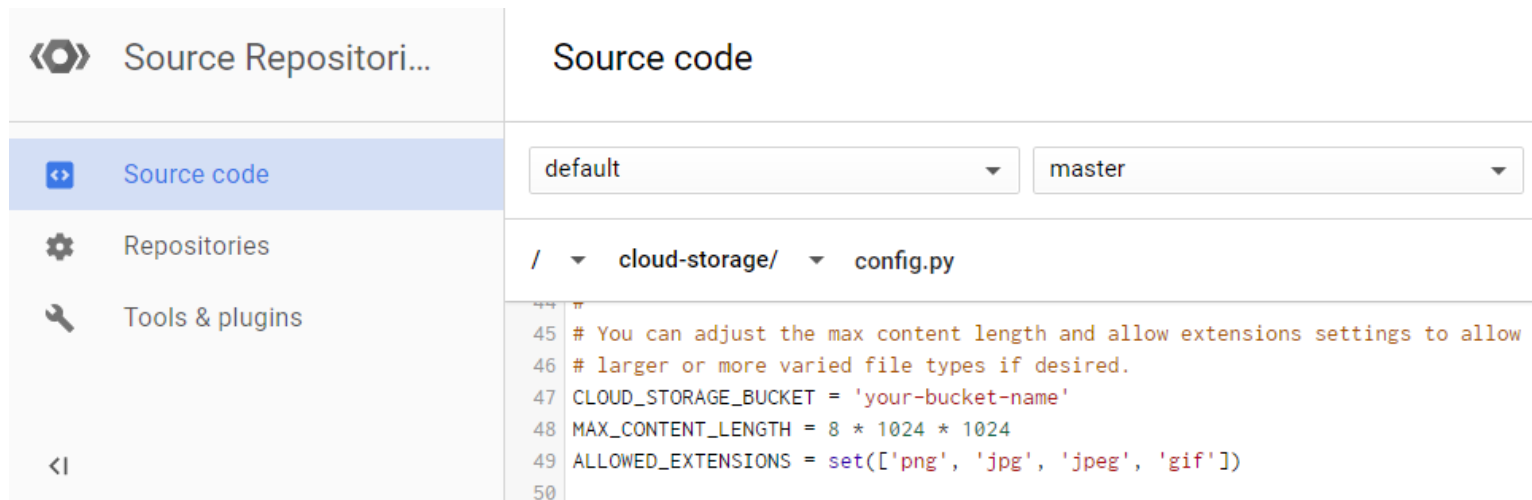
- Run within your class project (not cp100)
- Via gcloud SDK (access via Google Cloud Shell or via linuxlab)

```
gsutil mb -l <location> gs://$DEVSHELL_PROJECT_ID
```

- `gsutil` (Google Cloud Storage utility) command
- `mb` = make bucket
- Use `<location>` of `us-west1`
- `gs://`
 - URI for all buckets (must be globally unique)
 - Use `<Project ID>` to uniquely label bucket
 - Note: you can use any name that is unique but the instructions assume you've named your bucket after your project ID
 - Get Project ID in Google Cloud Shell via
- Verify in console that bucket has been created
- Allow global read access to bucket

```
gsutil defacl ch -u AllUsers:R gs://$DEVSHELL_PROJECT_ID
gsutil defacl set public-read gs://$DEVSHELL_PROJECT_ID
```

- App located in source repository from previous lab within `cloud-storage` directory
 - Examine `config.py` to see bucket name configuration and allowed filename extensions for image uploads



The screenshot shows a web interface for a source code repository. On the left is a sidebar with navigation links: 'Source Repositori...' (with a hex icon), 'Source code' (highlighted with a blue bar and a code icon), 'Repositories' (with a gear icon), and 'Tools & plugins' (with a wrench icon). The main area is titled 'Source code' and contains two dropdown menus: 'default' and 'master'. Below these is a breadcrumb path: '/ > cloud-storage/ > config.py'. The code editor displays the following Python code:

```
45 # You can adjust the max content length and allow extensions settings to allow
46 # larger or more varied file types if desired.
47 CLOUD_STORAGE_BUCKET = 'your-bucket-name'
48 MAX_CONTENT_LENGTH = 8 * 1024 * 1024
49 ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg', 'gif'])
50
```

- Examine `bookshelf/storage.py` to see code for writing to bucket
 - Cloud Storage URI is returned so database can set `imageUrl` property
 - Filename created with timestamped to avoid naming conflicts

Source code

default

master

/ ▾ cloud-storage/ ▾ bookshelf/ ▾ storage.py

```

44 # [START upload_file]
45 def upload_file(file_stream, filename, content_type):
46     """
47     Uploads a file to a given Cloud Storage bucket and returns the public url
48     to the new object.
49     """
50     _check_extension(filename, current_app.config['ALLOWED_EXTENSIONS'])
51     filename = _safe_filename(filename)
52     bucket = current_app.config['CLOUD_STORAGE_BUCKET']
53
54     objectname = '/' + bucket + '/' + filename
55
56     cs_file = cloudstorage.open(objectname,
57                                'w',
58                                content_type=content_type)
59     cs_file.write(file_stream)
60     cs_file.close()
61
62     url = 'https://storage.googleapis.com' + objectname
63
64     if isinstance(url, six.binary_type):
65         url = url.decode('utf-8')
66
67     return url
68 # [END upload_file]

```





/ ▾ cloud-storage/ ▾ bookshelf/ ▾ storage.py

```

31 def _safe_filename(filename):
32     """
33     Generates a safe filename that is unlikely to collide with existing objects
34     in Google Cloud Storage.
35
36     ``filename.ext`` is transformed into ``filename-YYYY-MM-DD-HHMMSS.ext``
37     """
38     filename = secure_filename(filename)
39     date = datetime.datetime.utcnow().strftime("%Y-%m-%d-%H%M%S")
40     basename, extension = filename.rsplit('.', 1)
41     return "{0}-{1}.{2}".format(basename, date, extension)

```

- Examine `bookshelf/crud.py` to see code for uploading and setting `imageUrl` property for book

 Source Repositori...	Source code
 Source code	default master
 Repositories	/ cloud-storage/ bookshelf/ crud.py
 Tools & plugins	<pre>62 @crud.route('/add', methods=['GET', 'POST']) 63 def add(): 64 if request.method == 'POST': 65 data = request.form.to_dict(flat=True) 66 67 # If an image was uploaded, update the data to point to the new image. 68 # [START image_url] 69 image_url = upload_image_file(request.files.get('image')) 70 # [END image_url] 71 72 # [START image_url2] 73 if image_url: 74 data['imageUrl'] = image_url 75 # [END image_url2] 76 77 book = get_model().create(data) 78 79 return redirect(url_for('.view', id=book['id'])) 80 81 return render_template("form.html", action="Add", book={})</pre>

- Examine `bookshelf/templates/list.html` to see code for displaying book when given a dict of books from model code

/ ▾ cloud-storage/ ▾ bookshelf/ ▾ crud.py

```
45 @crud.route("/")
46 def list():
47     token = request.args.get('page_token', None)
48     books, next_page_token = get_model().list(cursor=token)
49
50     return render_template(
51         "list.html",
52         books=books,
53         next_page_token=next_page_token)
```

/ ▾ cloud-storage/ ▾ bookshelf/ ▾ templates/ ▾ list.html

```
27 {% for book in books %}
28 <div class="media">
29     <a href="/books/{{book.id}}">
30         <div class="media-left">
31             {% if book.imageUrl %}
32                 
33             {% else %}
34                 
35             {% endif %}
36         </div>
37         <div class="media-body">
38             <h4>{{book.title}}</h4>
39             <p>{{book.author}}</p>
40         </div>
41     </a>
42 </div>
```

- In Console, `cd ~/cp100/default/cloud-storage` and edit `config.py` and edit it to point to your storage bucket (<projectID>)
 - Or use the `sed` command, but use
`sed -i s/your-bucket-name/$DEVSHHELL_PROJECT_ID/ config.py`
- Note that GCS libraries now needed in `requirements.txt`

/ ▾ cloud-storage/ ▾ requirements.txt

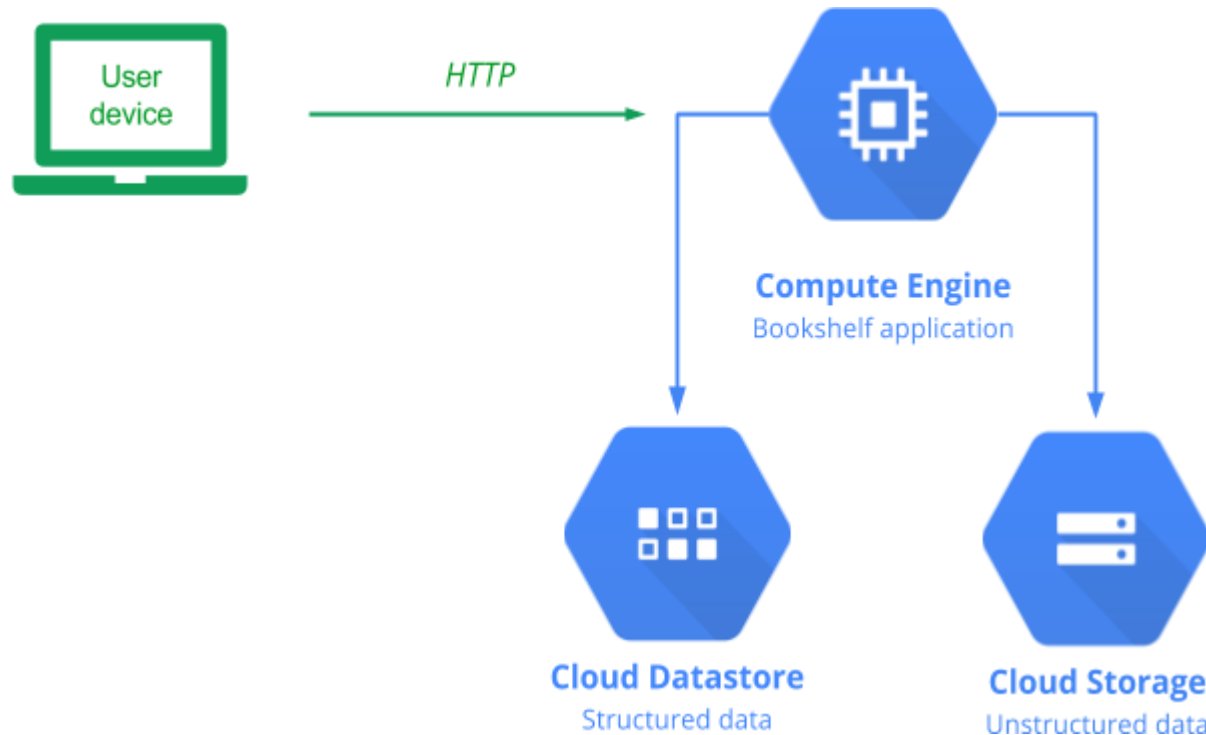
```
1 Flask==0.11.1
2 GoogleAppEngineCloudStorageClient==1.9.22.1
3 gunicorn==19.6.0
4 six==1.10.0
```

- Install requirements and deploy (see walkthrough)
- Download images and create book as instructed

- Turn in
 - Show book that is added in section 6 of walkthrough (CPD200..)
 - Show time-stamped image of book cover used in storage bucket via console

IaaS+DBaaS+Cloud Storage Lab #1

- Bookshelf app on Compute Engine (30 min)



- Changes
 - Uses Cloud Datastore and Cloud Storage directly (instead of from App Engine)
 - Small changes in client library to migrate from App Engine PaaS to unmanaged version on an IaaS model

- **Bring up instance in Cloud Shell**

- Set zone to `us-west1-b`
- Run command
- Run command

```
gcloud compute instances create bookshelf \  
  --image-family=debian-8 --image-project=debian-cloud \  
  --machine-type=g1-small \  
  --scopes userinfo-email,cloud-platform \  
  --metadata-from-file startup-script=startup-scripts/startup-  
    script.sh \  
  --tags http-server
```

- Image type to debian, machine type to small, binds owner of instance
- Specifies startup script to run upon launch
- Tags with label that allows HTTP traffic through the firewall to the instance

- Add additional firewall rule to http-server tag since non-standard port is used (8080)

```
gcloud compute firewall-rules create default-allow-http-8080 \  
  --allow tcp:8080 \  
  --source-ranges 0.0.0.0/0 \  
  --target-tags http-server \  
  --description "Allow port 8080 access to http-server"
```

- Adds rule to http-server tag that allows traffic to TCP port 8080 from any source (0.0.0.0/0)

- Startup script
 - Bring up environment onto initial vanilla VM
 - Done manually in this lab
 - Automated tools for doing similar functions include Puppet, Ansible, Chef
 - Subsumed by Google Deployment Manager on GCP (but other tools can still be used)

• Startup script

/ ▾ compute-engine/ ▾ startup-scripts/ ▾ startup-script.sh

```
29 # Install dependencies from apt
30 apt-get update
31 apt-get install -yq \
32     git build-essential supervisor python python-dev python-pip libffi-dev \
33     libssl-dev
34
35 # Create a pythonapp user. The application will run as this user.
36 useradd -m -d /home/pythonapp pythonapp
37
38 # pip from apt is out of date, so make it update itself and install virtualenv.
39 pip install --upgrade pip virtualenv
40
41 # Get the source code from the Google Cloud Repository
42 # git requires $HOME and it's not set during the startup script.
43 export HOME=/root
44 git config --global credential.helper gcloud.sh
45 git clone https://source.developers.google.com/p/$PROJECTID /opt/app
46
47 # Replace the project ID placeholder in config.py
48 sed -i s/your-project-id/"$PROJECTID"/ /opt/app/compute-engine/config.py
49
50 # Install app dependencies
51 virtualenv /opt/app/compute-engine/env
52 /opt/app/compute-engine/env/bin/pip install -r /opt/app/compute-engine/requirements.txt
```

/ ▾ compute-engine/ ▾ requirements.txt

```
1 Flask==0.11.1
2 gcloud==0.18.1
3 gunicorn==19.6.0
4 six==1.10.0
5 honcho==0.7.1
6
```

- Examine code

- Source Repositories=>Source code=>compute-engine
- Note: Alternative client libraries used to access Cloud Datastore for Compute Engine version versus App Engine

/ ▼ app-engine/ ▼ bookshelf/ ▼ model_datastore.py

```
16 from google.appengine.datastore.datastore_query import Cursor
17 from google.appengine.ext import ndb
74 def read(id):
75     book_key = ndb.Key('Book', int(id))
76     results = book_key.get()
77     return from_datastore(results)
```

/ ▼ compute-engine/ ▼ bookshelf/ ▼ model_datastore.py

```
16 from gcloud import datastore
26 def get_client():
27     return datastore.Client(current_app.config['PROJECT_ID'])
58 def read(id):
59     ds = get_client()
60     key = ds.key('Book', int(id))
61     results = ds.get(key)
62     return from_datastore(results)
```


- Link: Bookshelf app on Compute Engine (30 min)
 - <https://codelabs.developers.google.com/codelabs/cp100-compute-engine/>