Pre-Requisites

- AWS Free Tier Account
- Software:
  - Docker
- Libraries:
  - pep8
  - pep257
  - unittest2
  - pytest
  - bottle
  - cherrypy<9.0
- Internet Connection

Basic Pillars of ML deployment

What is ML deployment?

Scalable inference on the cloud, in a timeframe that allows superior **user experience**, while being **cost-efficient.**

MLOps is the new cutting edge skill.

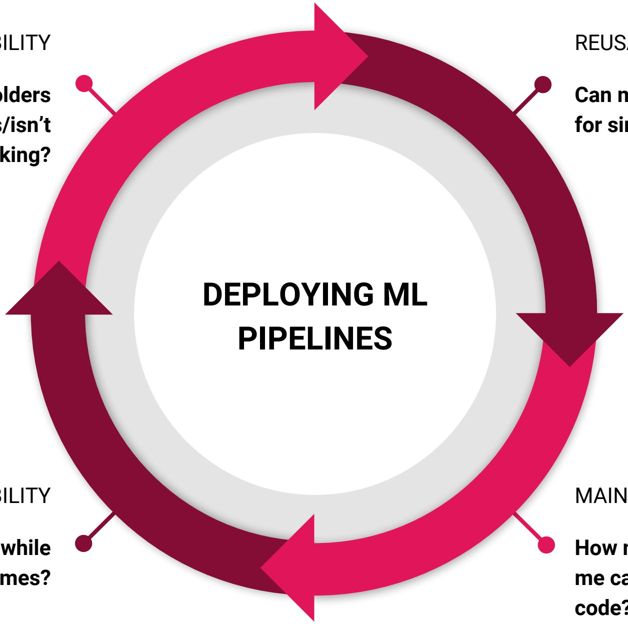
Building AND deploying your own algorithm/models avoids “chinese whispers”

Trouble with Deploying ML

- Why is this slow?
- Why is this hogging all the CPU?
- Where's all the memory gone?
- What's with all these error messages?
- **Classic -** "It works on my computer"

And many more ….

4 Pillars of ML Deployment



EXPLAINABILITY

Can I explain to stakeholders why my model is/isn’t working?

REUSABILITY

Can my codebase be reused for similar projects?

SCALABILITY

Does my code falter while processing volumes?

MAINTAINABILITY

How many people other than me can understand my code?

Reusability

- ❑ Are their “duplications” in the codebase?
- ❑ Is the code modular?
- ❑ Are their redundant functions/variables ?
- ❑ How many variables are hardcoded?
- ❑ Are their too many heuristics involved?
- ❑ Is there a configuration file to manage variables?
- ❑ Are you reinventing the wheel?
- ❑ Did you copy-paste that GH/SO code ?

Code reusability often separates the amateurs from the professionals.

Reusing code not only makes most sense when building a project, but it adds to the quality of your code. Building code for reusability often takes practice, but is easily achieved with some discipline while programming.

Reusability - POP QUIZ 1

You need to create a logistic regression model on house prices, and you want to impress your boss. What do you do?

a. Recode LR from scratch in Numpy
b. Use Scikit for LR implementation, and work on feature engineering instead

Reusability - POP QUIZ 2

Which function is better written for reusability?

```
stopchars = [a, b, c, d]

def  normalize_word(word):
        word =  word.strip().lower()
        word = “”.join([char for char in word if char not in stopchars])
        return word
```

```
stopchars = [a, b, c, d]

def  normalize_word(word, stops=stopchars):
        word =  word.strip().lower()
        word = “”.join([char for char in word if char not in stops])
        return word
```

Maintainability

- ❏ How many lines of code?
- ❏ Are their “duplications” in the codebase?
- ❏ Missing docstrings/code comments?
- ❏ PEP8/PEP257 compliant?
- ❏ Do my function names explain themselves?
- ❏ Are their redundant functions/variables ?

Maintaining a codebase is part of the role for an ML Engineer.

What very few understand is that the technical debt and pain associated with refactoring a code could have been avoided with very simple methods early on.

Using tools like CodeClimate, PyCharm helps you avoid this trap.

Maintainability - POP QUIZ 3

What’s a better function name for a function that normalizes a word token?

a. **preprocess_word**
b. clean_word
c. **process_token**
d. **process_string**
e. **normalize_word**

Maintainability - POP QUIZ 4

Which function is better written?

```
def  normalize_word(word):
        return word.strip().lower()
```

```
def  normalize_word(word):
        “””Processes and returns a word, stripping
whitespaces and lowercases it”””

        return word.strip().lower()
```

Scalability

- ❏ How many complex data structures in the codebase?
- ❏ Is the code dependent on local environment variables ?
- ❏ Is the code optimized for search ?
- ❏ What's the test coverage percentage?
- ❏ Do the test cases include rarely encountered edge cases?
- ❏ Does the code execute CPU or memory intensive tasks?
- ❏ Has the code been optimized with data structures and algorithms?
- ❏ Are you caching results ?
- ❏ Have you profiled your functions?

Scalability of a codebase means very different than the scalability of an application.

Many a times, a certain component of a codebase becomes a bottleneck, which slows the complete pipelines. Writing scalable code often requires a good understanding of data structures and techniques like memoization

Explainability

- ❏ Do you understand the code’s behaviour?
- ❏ How *readable* is your code?
- ❏ Is your model linear/non-linear
- ❏ Have your ML feature importances been mapped?
- ❏ Are you using frameworks like ELI5 or LIME?
- ❏ Are your features independent?
- ❏ Is your model training data well balanced?
- ❏ Do you have knowledge of the environment variables that may affect predictions?

Explainability of your code and your ML are both important, and can be not-so-straightforward at times.

If you are using a non-linear model, explainability becomes very difficult. For decision trees, visualizing features importances give an insight on how to improve the models. Ex. LIME, ELI5

Writing Production Grade code

S02

The Programmer from Hell

- Variable names don’t make sense
- No respect for indentation
- Mixed up indentation
- No comments for empirical/business logic steps
- No docstrings
- No usage examples
- No test cases
- PEP8/PEP257 non-compliant

Bad Code doesn’t mix well with prod

- Maintenance issues
- Technical Debt
- Upgrades start breaking stuff
- Backward compatibility is hit
- Prod rollouts are delayed
- Programming progress is hindered
- Code base is polluted
- No uniform standard in team

The philosophy

"A universal convention supplies all of maintainability, clarity, consistency, and a ***foundation for good programming habits*** too.

What it doesn't do is insist that you follow it against your will. That's Python!"

—Tim Peters on comp.lang.python, 2001-06-16

PEP8

Deals with how to ***style your python code.*** It defines the overall philosophy of structuring your code in an “easy-on-my-eyes” manner

3 Major Takeaways:

- Naming Conventions
- Code Comments
- Code Styling

PEP8 - Naming Conventions

- Function names should be lowercased, with words separated by underscores as necessary to improve readability
- Class names should start with an upper case
- Avoid reserved keywords as variables/function names
- Never use single character variable names
- For Constants, use all capital letters with underscores separating words
- Avoid function/variable naming similar/same to libraries

PEP8 - Code Comments

- Comments should be complete sentences.
- Block comments must have same indentation as the following code
- Write docstrings(PEP257) for all public modules, functions, classes, and methods
- Use inline comments sparingly, but must for business/empirical logic
- Inline comments should be separated by at least two spaces from the statement

PEP8 - Code Styling

- Focus on the readability of the code - other team members must be able to understand your code properly
- Use **is not** operator rather than **not ... is**
- Prefer **def** statement than ad-hoc **lambda** statements
- When catching exceptions, mention specific exceptions whenever possible instead of using a bare **except**
- Object type comparisons should always use isinstance() instead of comparing types directly
- Don't compare boolean values to True or False using ==, instead use **if X**

PEP257

Deals with ***docstring conventions***, and help standardize docstring structures

DOCSTRING:

A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition.

Broad strokes of PEP257

- The docstring should summarize its behavior and document its arguments, return value(s), side effects, exceptions raised, and restrictions on when it can be called
- Stick to triple quotes for single/multiline comments
- No blank line either before or after the docstring
- Multi-line docstrings consist of a summary line just like a one-line docstring, followed by a blank line, followed by a more elaborate description.

Staying Compliant

- Use IDEs - continuous checking for format, redundancies, quick cleanups
- Stick to one style of indentation (spaces OR tabs)
- Use pylint to highlight issues
- Use native PEP8/PEP257 packages to check for potential issues
- Online tools to do a quick review of your code
- In the end, focus on making sure that your code ***looks clean***, not a hot mess

Tests for your code - WHY

- Essentially **assert** statements
- Check for code consistency and behaviour
- Increasing code coverage reduce chances of future build failing
- Writing unit tests while you are writing your code - recommended
- Helps refine your codebase and improve quality
- Debugging becomes faster
- Instills better understanding of the code itself for the programmers

Tests - HOW

- A testing unit should focus on one tiny bit of functionality and prove it correct
- Each test unit must be fully independent. Each test must be able to run alone, and also within the test suite, regardless of the order that they are called
- Tests should run fast and not rely on overtly complex setup/data structures
- Always run the full test suite before a coding session, and run it again after
- It is a good idea to implement a hook that runs all tests before pushing code to a shared repository - Ex. Travis CI
- Use long and descriptive names for testing functions

Reference: http://docs.python-guide.org/en/latest/writing/tests/

Testing Frameworks

- **unittest** is the batteries-included test module in the Python standard library
- **unittest2** is a backport of Python 2.7's unittest module which has an improved API and better assertions over the one available in previous versions of Python
- **Pytest** is a no-boilerplate alternative to Python's standard unittest module
- Unittest/2 is lot more extensible and mature, but requires a structured thinking to writing tests
- You can also dynamically generate test cases - allows detection of edge cases pretty quickly

Hands on Tutorial 1

- With a piece of provided code, refactor it & improve its PEP8/PEP257 standing
- Suggest a list of tests to increase the code coverage
- Code to be provided by the instructor

Hands on Tutorial 2

- With your own code from a previous GreyAtom exercise, refactor it & improve its PEP8/PEP257 standing
- Write a list of tests to increase the code coverage
- Code to be provided by the instructor

Deploying ML

S03

Why Deploy your ML codebase

- Open Source contribution
- Earn creds
- Company/Consulting project
- Academic requirements

How to Deploy your ML codebase

- Release as a PyPi package
- Deploy on the cloud via ML products
- Deploy as an API
- Package the complete application as a Dockerfile

PyPi

- The Python Package Index (PyPI) is a repository of software for the Python programming language
- Over a 130K projects in the repository
- Most popular way to distribute OSS
- But, also home to many malicious software

WHEN

- Kicking off an OSS in a specific use-case, with plans for future contributions from users.

Cloud Infrastructure

- Services like AWS Sagemaker, Google ML, Azure ML
- You upload the data, auto optimization and prediction via an API
- Easier to manage versions and easy rollbacks
- You pay per prediction
- Becomes expensive over a long term

WHEN

- You don’t have a lot of data scientists to iterate, limited knowledge of algorithms and scalability is a key

API

- Deploy your ML model via Bottle/Flask
- You upload the data, self optimize and expose prediction via an API
- Tools like Postman, Locust to test the API’s outputs and load bearing capacity
- You pay for the underlying computing resources
- Non-scalable, yet perfect way to test a new model with domain experts

WHEN

- You are building a demo or a sandbox to test a hypothesis for a limited set of users
-

Docker

- Deploy your ML model but packaged with Docker
- You upload the data, self optimize and expose prediction via an Docker entrypoint
- Tools like AWS ECR, ECS, Kubernetes allow you to manage scaling easily
- You pay for the underlying computing resources
- Highly scalable, and best method to deploy ML in production

WHEN

- You are rolling out your ML codebase for a wider audience
-

Docker

S04

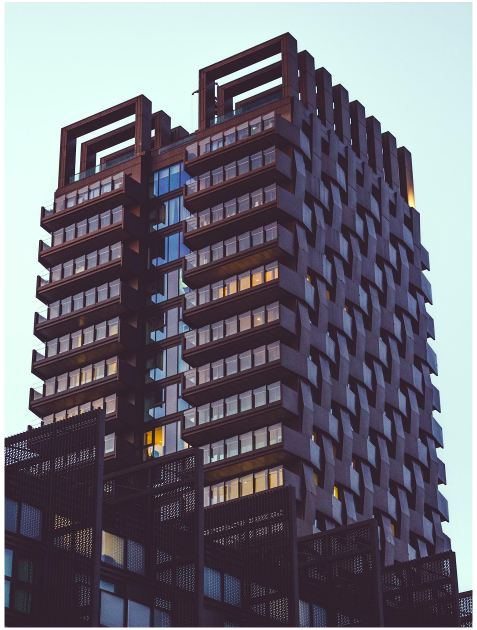
What are containers

- “Lightweight VMs” - But is it really?
- Not virtual machines, but app-delivery model mechanism

“In a VM-centered world, the unit of abstraction is a monolithic VM that **stores not only application code, but often the stateful data**. A VM takes everything that used to sit on a physical server and just packs it into a single binary so it can be moved around. But it is still the same thing.

With Docker containers the **abstraction is the application**; or more accurately a service that helps to make up the application.“

Docker vs VM



Before Docker

- Setting up application environment was messy
- “It works on my machine” - Really?
- Application rollout was difficult and long winded
- Replicability was not the primary concern
- Scaling applications required a host of developers
- Too many moving parts while working with VMs
- CI/CD was clunky and non-reliable

Why Docker

As Martin Fowler puts it, Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.

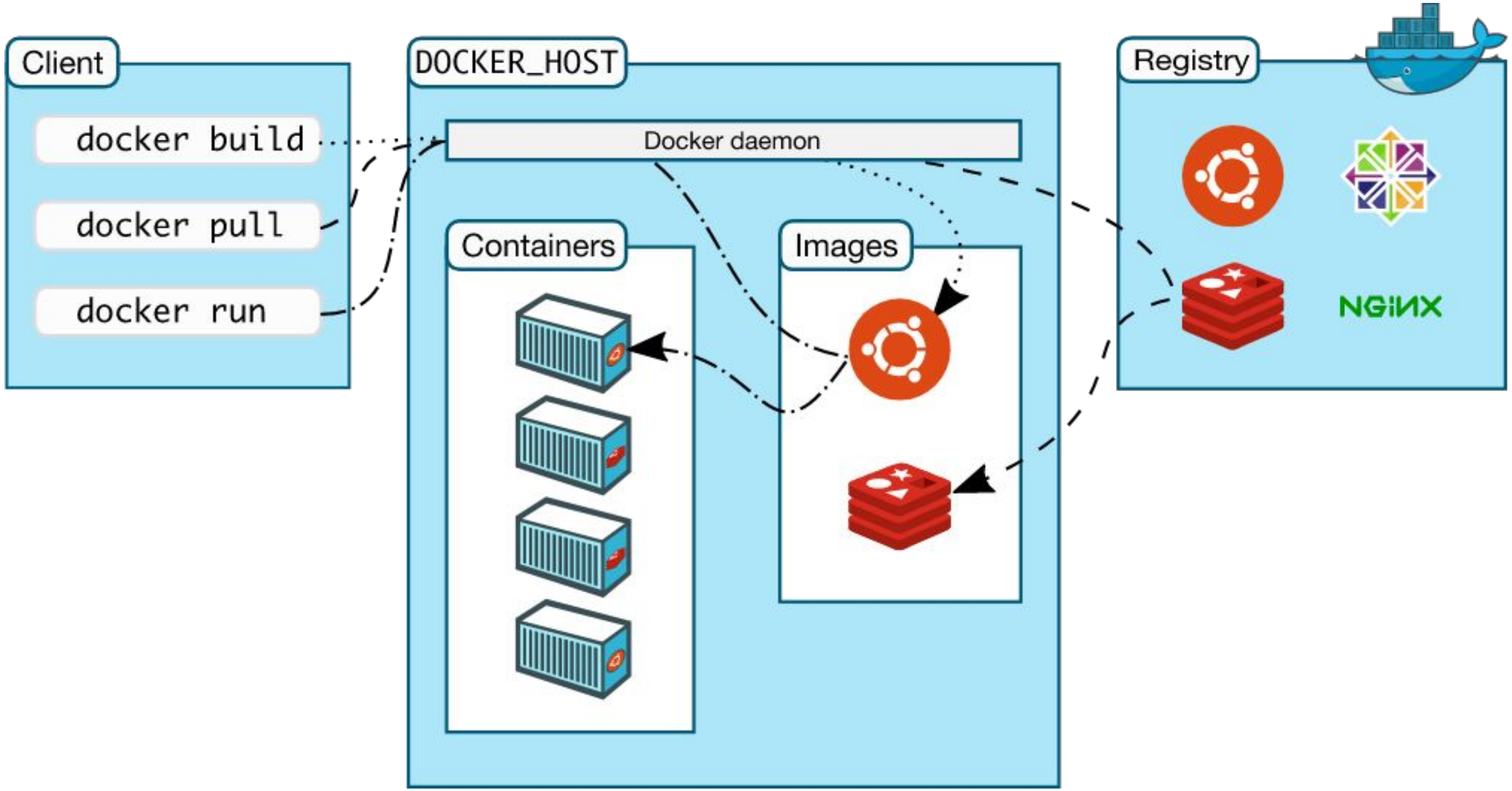
You actually do continuous delivery if:

- Your software is deployable throughout its lifecycle.
- Your team prioritizes keeping the software deployable over working on new features.
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them.
- You can perform push-button deployments of any version of the software to any environment on demand

Development with Docker

- Create requirements for an application within a Dockerfile or use Docker as a remote interpreter
- Push the code up to git repo
- A CI system builds the same environment that used in development to run the tests in prod
- Roll out the exact configuration used in dev, testing and staging before prod
- Be happy

Brief Overview: Docker Architecture



Docker Terminology

CONTAINER:

A container is a runtime instance of a docker image.

A Docker container consists of

- A Docker image
- An execution environment
- A standard set of instructions

The concept is borrowed from Shipping Containers, which define a standard to ship goods globally. Docker defines a standard to ship software.

Docker Terminology

BASE IMAGE:

An image that is built upon with systematic filesystem changes to build the final image

LAYERS:

Layers are applied in sequence to the base image to create the final image. When an image is updated or rebuilt, only layers that change need to be updated, and unchanged layers are cached locally. This is part of why Docker images are so fast and lightweight. The sizes of each layer add up to equal the size of the final image.

IMAGE

Docker images are the basis of containers. An Image is an **ordered collection of root filesystem changes (layers)** and the corresponding execution parameters for use within a container runtime. An image does not have state and it never changes unless rebuilt.

Docker Commands

docker build :

Build a docker image from a Dockerfile

docker run:

Run a docker container with some specified variables

docker exec

Execute a command inside a running docker container

Docker Commands

docker rm

Remove one or more containers

docker ps

List all the containers

docker images

List all the docker images on the local machine

Docker File - Best practices

- Understand “WHY” you are building the Docker image
- Minimize the layers and dependencies
- Sort installs and arguments to avoid duplicates
- Use cache to speed up docker builds
- Comment your Dockerfile to help understand steps
- Avoid “sudo”
- Use the best possible image from registry to minimize reinvention
- Think ephemeral, minimalism is the key. Be a “Gangster”

Docker File - Commands

FROM

Defines the base image

LABEL

Add metadata for your app

RUN

Build time command to add and configure layers

Docker File - Commands

CMD

Command to run during runtime

EXPOSE

Specify open ports of the containers during runtime

ENV

Define path variables for the container during buildtime to use during runtime

Docker File - Commands

ADD/COPY

Copy files from host during build time

WORKDIR

Specify working directory during build/runtime

ENTRYPOINT

Define default command for the docker container

Hands-on Tutorial 1

WHAT WE WILL DO

- Write a Dockerfile that installs PyTorch , Keras and TensorFlow
- Copy files from local that contain a sample code to print Keras code version
- Build the Docker image
- Mount and map a local folder to a running Docker container
- Use nano to modify a file to reflect locally
- Stop the Docker container

AWS

S05

Cloud Platforms

- Infrastructure platforms that can be used on a pay per use
- Leaders are AWS, MS Azure and Google Cloud
- Enables the deployment of business critical applications in a scalable environment
- OSI Model Layers 1-4 taken care of by the cloud providers
- 99.9% SLA, limited downtime and managed scalability
- Allows ML developers fastrack their DevOps capabilities

AWS

- AWS began offering its technology infrastructure platform in 2006
- Offers all the top requirements to satisfy security and audit requirements
- Recognized independent third-party attestations, reports and certifications
- Global infrastructure (including Mumbai) for edge deployments
- Resize and rescale your environment at a click of the button
- Low cost executions based on events - AWS Lambda

AWS - S3

- S3 is a **simple storage service** that offers software developers a highly-scalable, reliable, and low-latency data storage infrastructure at very low costs.
- Amazon S3 is also designed to be highly flexible. Store any type and amount of data that you want; read the same piece of data a million times or only for emergency disaster recovery; build a simple FTP application, or a sophisticated web application such as the Amazon.com retail web site
- Dropbox, Netflix were one of the previous clients of AWS S3
- 100 GB of data costs ~3 USD per month
- Allows you to secure data both at rest and transit very easily
- Three models of data storage - S3, S3-IA and Glacier

FAQS

AWS - S3

- **Amazon S3 Standard - Infrequent Access (Standard - IA)** is an Amazon S3 storage class for data that is accessed less frequently, but requires rapid access when needed. A has a thinner front end that provides nine one-hundredths of a percent less availability than S3 Standard

- **Amazon Glacier** stores data for as little as $0.004 per gigabyte per month. To keep costs low yet suitable for varying retrieval needs, Amazon Glacier provides three options for access to archives, from a few minutes to several hours. Some examples of archive uses cases include digital media archives, financial and healthcare records, raw genomic sequence data, long-term database backups, and data that must be retained for regulatory compliance.

AWS - EC2

- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable **compute capacity** in the cloud. It is designed to make web-scale computing easier for developers.
- Takes less than 5 minutes to spin up a server and get going
- Horizontal + Vertical scaling handled with extreme ease
- Charged per hour of instance usage
- Prepackaged AMIs for deep learning, CV, NLP and Linux flavours

FAQS

AWS - ECR

- Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.
- Amazon ECR uses Amazon S3 for storage to make your container images highly available & accessible
- Amazon ECR transfers your container images over HTTPS and automatically encrypts images at rest
- You can easily push your container images to Amazon ECR using the Docker CLI from your development machine, and Amazon ECS can pull them directly for production deployments.

FAQS

AWS - ECS

- Amazon Elastic Container Service (ECS) is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.
- Use containers as a building block for your applications by eliminating the need for you to install, operate, and scale your own cluster management infrastructure
- Schedule long-running applications, services, and batch processes using Docker containers.
- Amazon ECS maintains application availability and allows you to scale your containers up or down to meet your application's capacity requirements
- Specify which container images are to be deployed, the CPU and memory requirements, the port mappings, and the container links

FAQS

AWS - Batch

- AWS Batch manages compute environments and job queues, allowing you to easily run thousands of jobs of any scale using Amazon EC2
- Supports any job that can executed as a Docker container. Jobs specify their memory requirements and number of vCPUs.
- It avoids idling compute resources with frequent manual intervention and supervision.
- Optimized for batch computing and applications that scale through the execution of multiple jobs in parallel
- Deep learning, genomics analysis, financial risk models, Monte Carlo simulations, animation rendering, media transcoding, image processing, and engineering simulations are all excellent examples of batch computing applications.

AWS - Sagemaker

- Amazon SageMaker is a fully-managed service that enables data scientists and developers to quickly and easily build, train, and deploy machine learning models
- Jupyter notebooks are supported. You can persist your notebook files on an attached ML storage volume.
- Amazon SageMaker includes built-in algorithms for linear regression, logistic regression, k-means clustering, PCA, factorization machines, neural topic modeling, LDA, gradient boosted trees, seq2seq, time series forecasting, word2vec, and CV. **Amazon SageMaker supports your custom training algorithms provided through a Docker image adhering to the documented specification**.
- No fixed limits to the size of the dataset you can use for training models, you can specify the Amazon S3 location of your training data as part of creating a training job

Hands-on Tutorial 1

WHAT WE WILL DO

- Launching an EC2 instance on AWS
- SSH into and and install packages on the server
- Securing the server through security groups

Hands-on Tutorial 2

WHAT WE WILL DO

- Install Docker on EC2
- Download data from S3
- Upload data to S3
- Securing objects on S3

Hands-on Tutorial 3

WHAT WE WILL DO

- Create an docker image on EC2
- Push the docker image to ECR
- Pull an image from ECR, update and re push the image

Introduction to RESTful

S06

Agenda

- Introducing the web
- Understanding REST philosophy
- Introduction to APIs
- How to design and implement APIs
- Documenting APIs

Internet & the World Wide Web

Everyday, we use technology so revolutionary that it doesn’t even register a “WOW” anymore. What we are using is the pinnacle of human ingenuity and innovation.

Yes, I am talking of the internet and the WWW, which we use to like and post cat and doge memes.

Web is the “killer app” of the internet

“The net existed before; the web is something built *on top*”

““The internet is the network of networks, and the underlying platform that makes the web—along with other technologies—possible. So the **web is one of these technologies that’s sitting on top of the internet**; it’s one use of the internet.”

Defining the web

TERM	ACRONYM FOR	DESCRIPTION
URL	Uniform Resource Allocator	**Where is a certain information located?** Ex. https://economictimes.indiatimes.com
HTML	Hyper Text Markup Language	**How is the information structured?** Ex. Heading, Body, Paragraph
HTTP	Hyper text transfer Protocol	**How is the information requested/transferred?** Ex. GET, PUT, POST, DELETE

HTTP Responses

- 200 - All Okay !
- 30X - Resource redirection
- 4XX - Authentication based errors, mainly client side
- 50X - Server side errors

HTTP Methods

METHOD	DESCRIPTION
GET	Retrieve data
HEAD	*GET, without the body*. Useful for debugging and checking resource presence
POST	Submit data to process and return a result
PUT	Add data
DELETE	Delete data

HTTP Methods vs CRUD

METHOD	CRUD POLICY
GET	Read
POST	Create
PUT	Create / Update
DELETE	Delete

What is REST

REST (REpresentational State Transfer) is an architectural style for developing web services.

It is very well defined by Roy Fielding in his PhD thesis “Architectural Styles and the Design of Network-based Software Architectures”

WHY REST

REST is popular due to its **simplicity** and the fact that it builds upon existing systems and features of the internet's HTTP in order to achieve its objectives, as opposed to creating new standards, frameworks and technologies.

REST vs SOAP

REST takes a **resource-based approach** to web-based interactions. With REST, you locate a resource on the server, and you choose to either update that resource, delete it or get some information about it.

With SOAP, the client doesn't choose to interact directly with a resource, but **instead calls a service, and that service mitigates access to the various objects** and resources behind the scenes.

Creating RESTful APIs - Best practices

- Think about your API and resources with the **lens of a finite state machine**
- Identify your resources properly
- Define your “entities” carefully
- Develop your API incrementally - C > R> U > D
- Your APIs should *describe themselves*

Examples

GOOD:

apiurl/predict/model/v1

BAD

apiurl/predict/model?v=1

Examples

GOOD:

https://stripe.com/docs/api

https://developer.github.com/v3/

BAD

https://www.flickr.com/services/api/

Bottle for APIs

- Bottle is a fast, simple and lightweight
- Built-in HTTP development server and support for paste, fapws3, bjoern, gae, cherrypy or any other WSGI capable HTTP server
- Convenient access to form data, file uploads, cookies, headers and other HTTP-related metadata
- Fast and pythonic built-in template engine and support for mako, jinja2 and cheetah templates
- Easy install : **pip install bottle**

HandsOn Tutorial 1

Hello World with Bottle

- Create a bottle App
- Print “Hello world !” when the API is pinged
- Use requests library to ping the API

HandsOn Tutorial 2

Create an API that does the following

- Returns the count of a word
- Update the count of a word
- Delete a word
- Create a word and pass a default value

HandsOn Tutorial 3

- Testing your API via Postman
- Generating code for API testing via Postman

ASSIGNMENT

S07

Assignment

- Document the API from the last assignment with PEP8/PEP257
- Refactor the code wherever necessary
- Create a docker image of the API and dependencies bundled
- Push the Docker image to Dockerhub
- Launch an EC2 instance on AWS
- Deploy the docker API on a server
- Ping the server with CRUD requests