API Design

REST API Design

API

- Accept and respond with JSON
- Use nouns instead of verbs in endpoint paths(This is because our HTTP request method already has the verb)
    - GET retrieves resources.
    - POST submits new data to the server.
    - PUT updates existing data.
    - DELETE removes data.

ref: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

REST API Design

API

- Use logical nesting on endpoints
  - '/articles/:articleId/comments'
- Handle errors gracefully and return standard error codes
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
  - 500 Internal server error
  - 502 Bad Gateway
  - 503 Service Unavailable

ref: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

REST API Design

API

- Allow filtering, sorting, and pagination
  - '/employees?lastName=Smith&age=30'
- Maintain good security practices
- Cache data to improve performance
- Versioning our APIs

ref: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

How to Version a REST API?

- URI Versioning
  - http://api.example.com/v1
  - http://apiv1.example.com
  - http://api.example.com/accounts/v1/
- Versioning using Custom Request Header
  - Accept-version: v1
  - Accept-version: v2
- Versioning using “Accept” header
  - Accept: application/vnd.example.v1+json
  - Accept: application/vnd.example+json;version=1.0

ref: https://restfulapi.net/versioning/

Semantic Versioning(1.2.1)

API

- **Major version**. This number is incremented for incompatible, i.e., breaking changes, e.g., removing an existing operation
- **Minor version**. This number is incremented if a new version provides new functionality in a compatible manner, e.g., adding a new operation to an API or adding a new feature to an existing operation
- **Patch version** (also called Fix Version). This number is incremented for compatible bug fixes, e.g., changing and clarifying the documentation in an API contract or changing the API implementation to fix a logic error.

ref: https://microservice-api-patterns.org/patterns/evolution/SemanticVersioning.html#

Compatibility

API

The general aim is that clients should not be broken by a service updating to a new minor version or patch. The kinds of breakages under consideration are:

- Source compatibility: code written against 1.0 failing to compile against 1.1
- Binary compatibility: code compiled against 1.0 failing to link/run against a 1.1 client library. (Exact details depend on the client platform; there are variants of this in different situations.)
- Wire compatibility: an application built against 1.0 failing to communicate with a 1.1 server
- Semantic compatibility: everything runs but yields unintended or surprising results

ref: https://cloud.google.com/apis/design/compatibility

Backwards-compatible (non-breaking) changes

API

- Adding an API interface to an API service definition
- Adding a method to an API interface
- Adding an HTTP binding to a method
- Adding a field to a request message
- Adding a field to a response message
- Adding a value to an enum
- Adding an output-only resource field

ref: https://cloud.google.com/apis/design/compatibility

Artifactory: Deploy/Delivery

API

Service: Deploy & Delivery(Docker Image)

- API: External/Internal
- Versioning
- Backward Compatibility

Library: Delivery

- API: External/Internal
- Versioning
- Backward Compatibility

API In Action

Revisit: Tax Exercise

- URI Path
- API Version

API In Action

Revisit: Tax Exercise

- add new properties **netIncome** for Response body

API In Action

Revisit: Tax Exercise

- User have to send Request body with **expense**

API In Action

Revisit: Tax Exercise

- User need to know PersonalTax by send their name

Docker Versioning

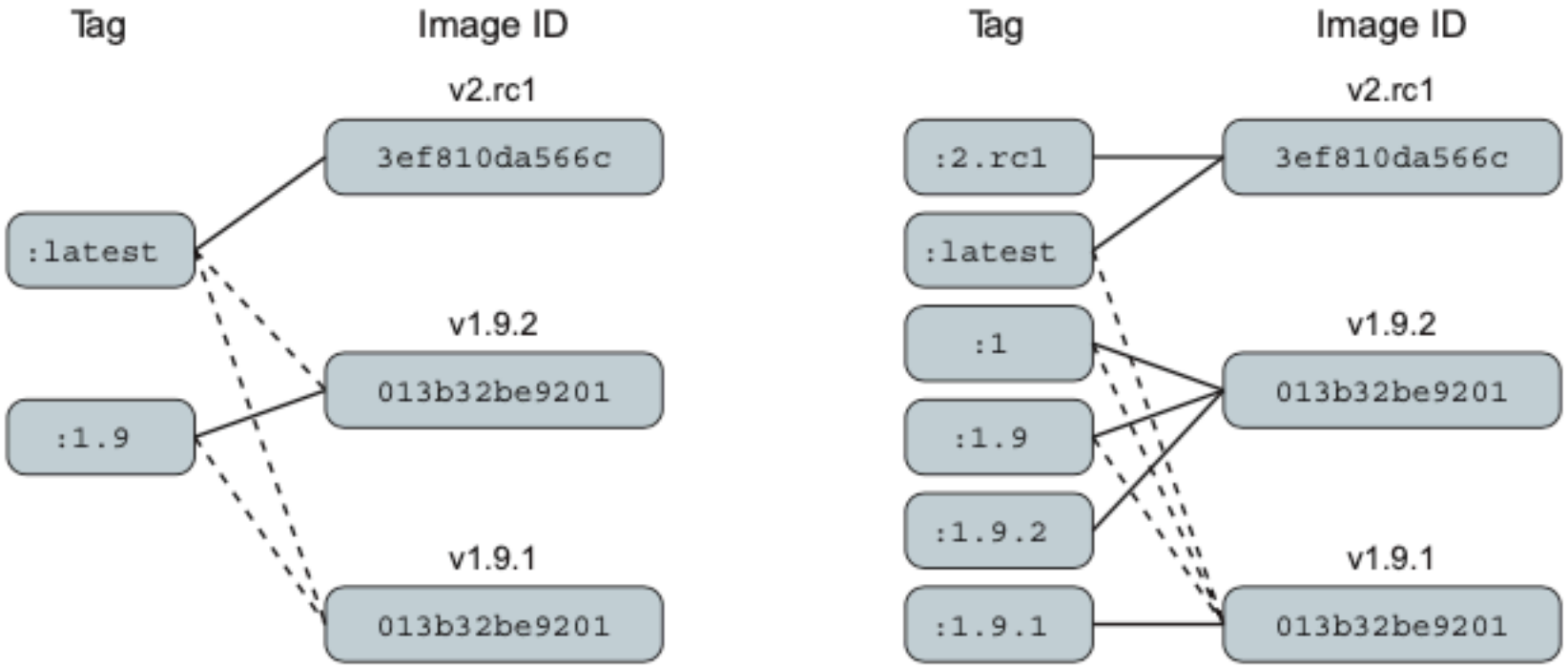

Figure 7.7 Two tagging schemes (left and right) for the same repository with three images. Dotted lines represent old relationships between a tag and an image.

- docker build -t app .
- docker build -t app:0.0.1 .