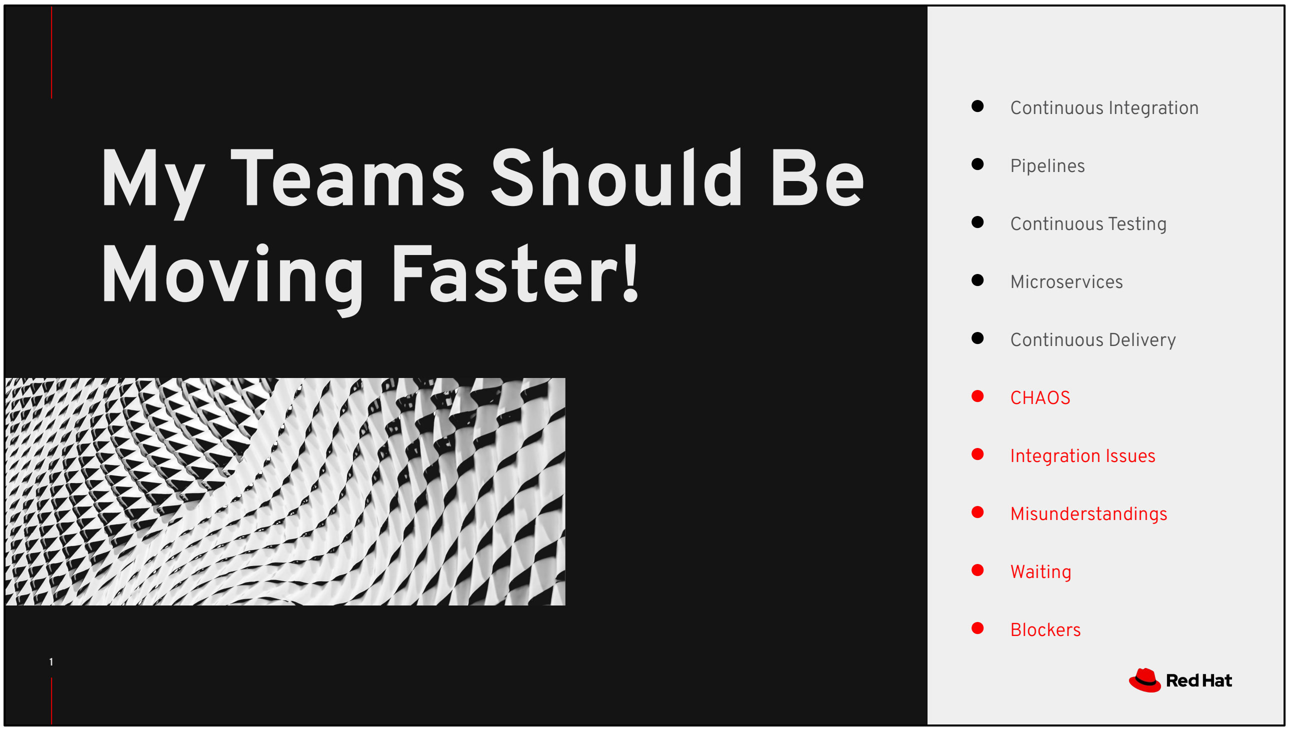


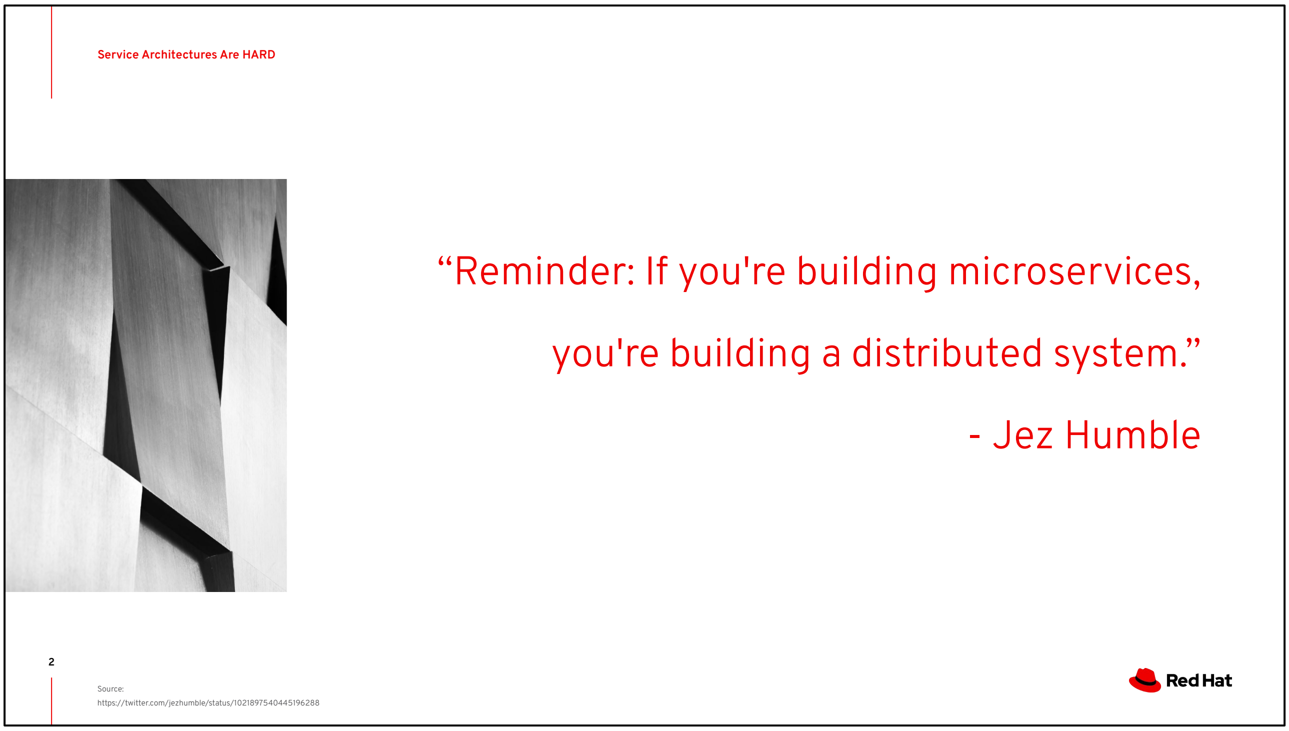
My Teams Should Be Moving Faster!

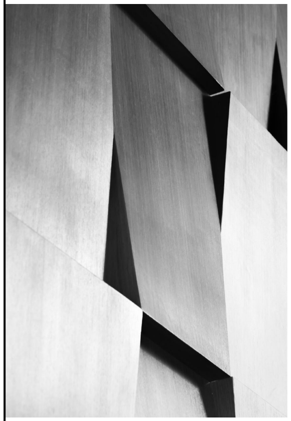
- Continuous Integration
- Pipelines
- Continuous Testing
- Microservices
- Continuous Delivery
- CHAOS
- Integration Issues
- Misunderstandings
- Waiting
- Blockers

You're adopting "DevOps" and "Microservices" because of the promise of delivering value faster to your users/customers. But you are not seeing the gains you want to see. You've got CI/CD/Pipelines/Tests/Microservices/etc… But you are still getting bogged down with teams being dependent on the progress of each other. Service A isn't done, so service B cannot work on integration. The UI requires access to ALL of the services, so it has to wait until the end to be integrated, and integration takes WEEKS because of back-and-forth issues between the frontend and backend and between different dependent services. This is NOT how this is supposed to work!

Many of us are familiar with writing code which other people interface with. In C & C++ we provide header files to tell other people what our code can do for them. In Java it might be an Interface definition. This works great for compile-time guarantees, but what do you do when you are writing services which work over the network or over the web?

If you have ever written an API or service which is consumed by others, even inside of your own team, this should be obvious. Every time you make a change to your service, those external and internal users are going to be annoyed because you just broke a BUNCH of their code. Take that annoyance and then consider what happens where there are multiple (perhaps multitudes of) services and worse yet when those APIs are consumed in the public by your customers. Add into that all of the complexities of operating a distributed system and you have a recipe for disaster.





"Reminder: If you're building microservices,

you're building a distributed system."

- Jez Humble

Source:
https://twitter.com/jezhumble/status/1021897540445196288

Red Hat

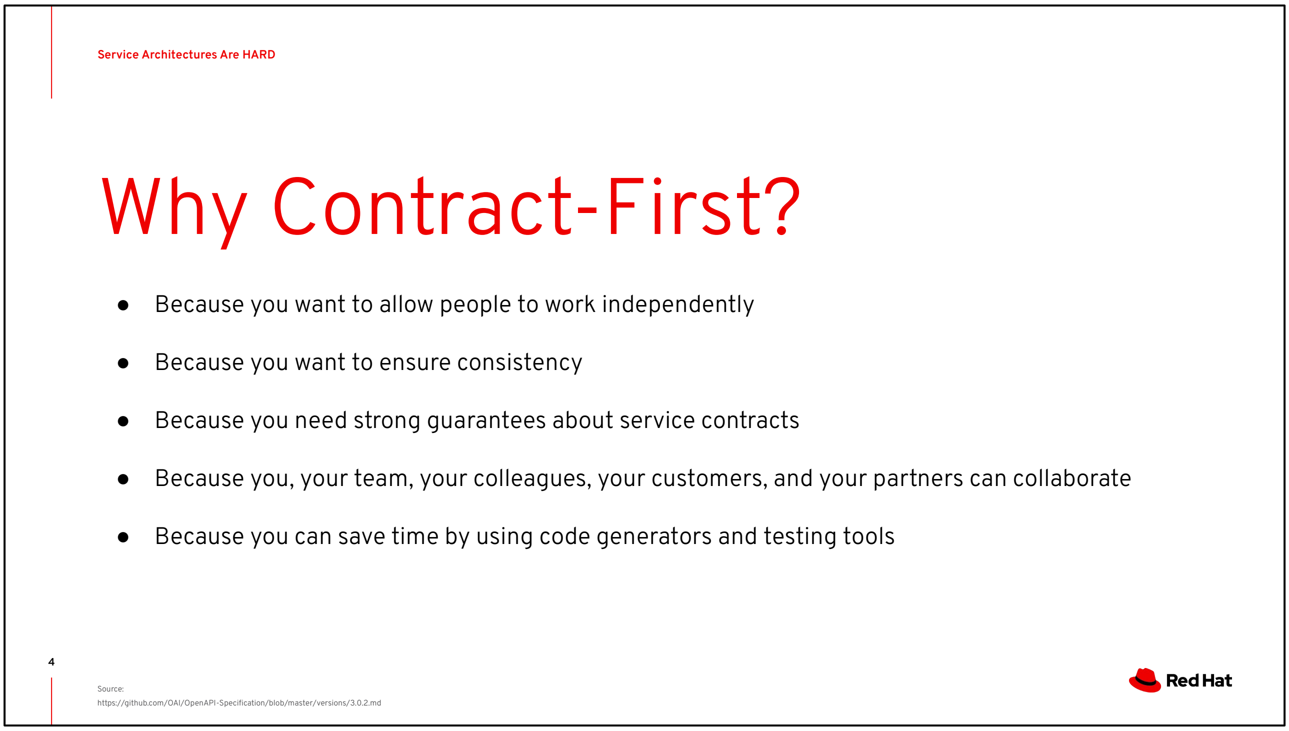
Lots of people have jumped on the Microservices bandwagon in recent years without TRULY understanding what that means. They have heard that their developer teams can achieve greater productivity, that their applications can achieve greater scalability, that their organizations can achieve greater agility, etc… What they fail to consider, in order to capitalize on those promises, is that they MUST fundamentally change how their teams/customers/consumers/partners work together. JUST implementing microservices is NOT going to achieve those goals because you are now designing a distributed system, and those components all have to work in a coordinated fashion. If you do not have a way for those disparate groups to coordinate with one another asynchronously, then you still have a major bottleneck. Your back-end devs are waiting on Database schemas and provisioning, your front-end devs are awaiting an API to code against, your customers and partners are awaiting documentation of those APIs in order to integrate with your services, etc.…

Red Hat
Application Services

Why You Should Be Doing Contract-First API Development

Deven Phillips
Senior Architect
@infosec812@foojay.social
https://github.com/infosec812

Red Hat



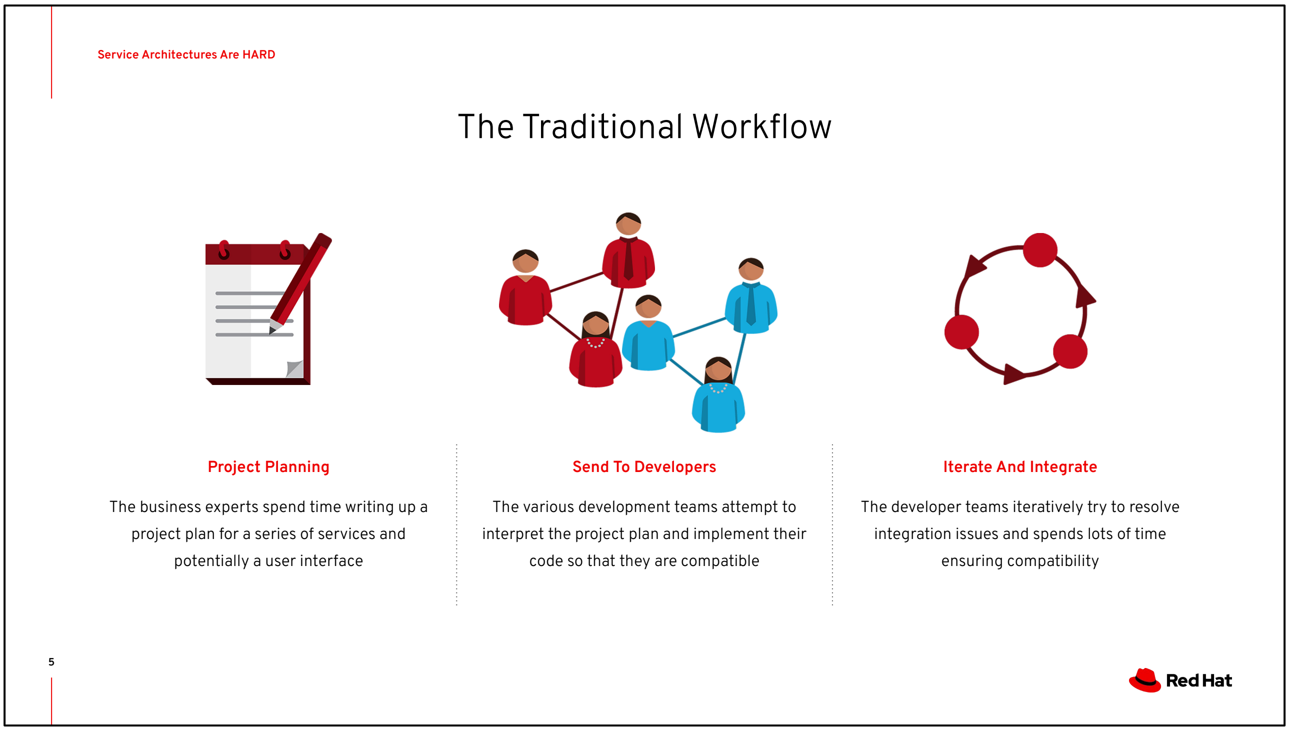
Why Contract-First?

- Because you want to allow people to work independently
- Because you want to ensure consistency
- Because you need strong guarantees about service contracts
- Because you, your team, your colleagues, your customers, and your partners can collaborate
- Because you can save time by using code generators and testing tools

Source:
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md

Red Hat

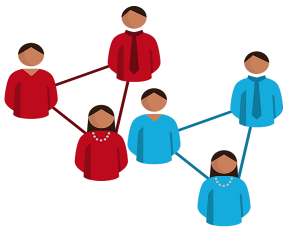
Regardless of if you find the code generation and tooling useful, having a simple format for sharing the contracts for your services amongst all of the users of that service is of significant value. Let’s learn a little about how simple and expressive it is to write OpenAPI specifications.



The Traditional Workflow

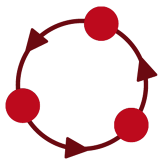
Project Planning

The business experts spend time writing up a project plan for a series of services and potentially a user interface



Send To Developers

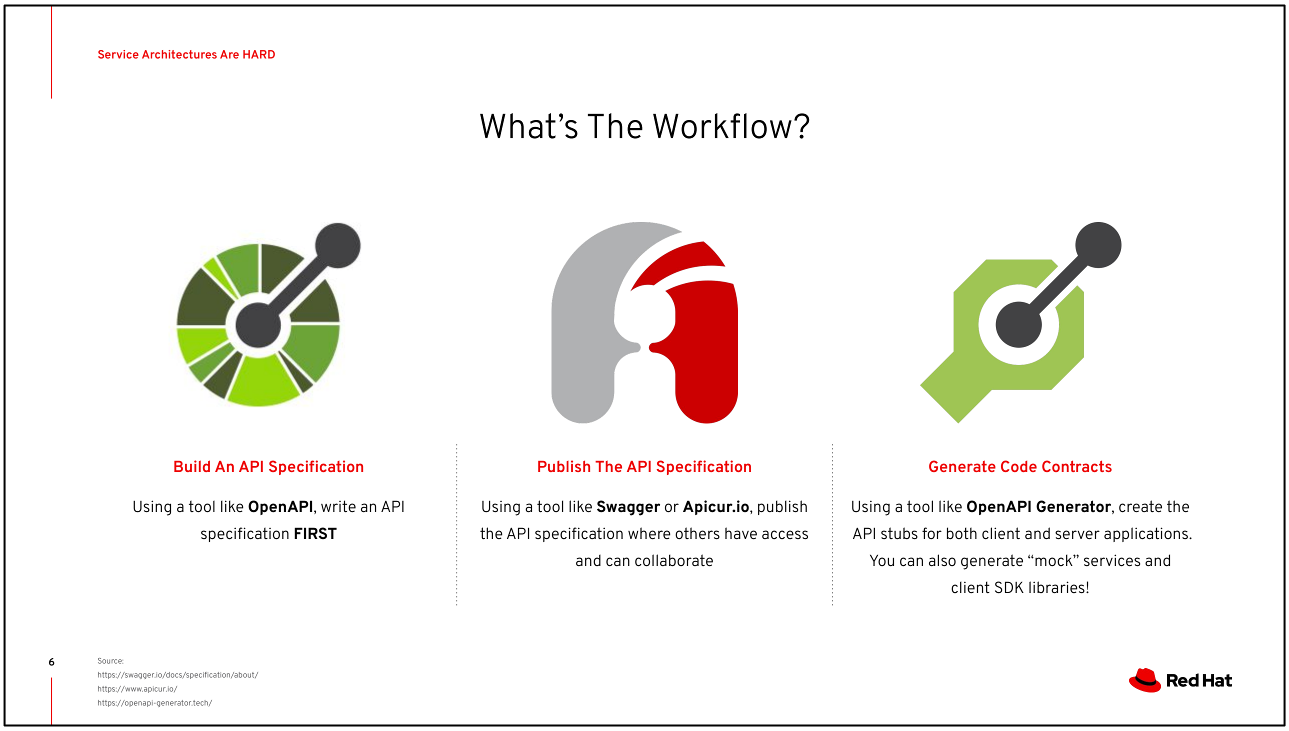
The various development teams attempt to interpret the project plan and implement their code so that they are compatible



Iterate And Integrate

The developer teams iteratively try to resolve integration issues and spends lots of time ensuring compatibility

This is a lot like traditional waterfall development of products, and as such it has lots of bottleneck which prevent efficient progress. The project planning is a time consuming process and often creates documents that are too vague and leave too much room for interpretation. The developers try to consistently implement code which is defined in the project plan, but differences of understanding and terminology make it easy for separate teams to end up with incompatible implementations. This results in lots of time and iterations required to resolve compatibility and integration issues at a point very far along in the development process. Lots of re-work, debugging, miscommunications, etc… Wouldn't it be better if we could get faster feedback?



What's The Workflow?

Build An API Specification

Using a tool like **OpenAPI**, write an API specification **FIRST**

Publish The API Specification

Using a tool like **Swagger** or **Apicur.io**, publish the API specification where others have access and can collaborate

Generate Code Contracts

Using a tool like **OpenAPI Generator**, create the API stubs for both client and server applications. You can also generate "mock" services and client SDK libraries!

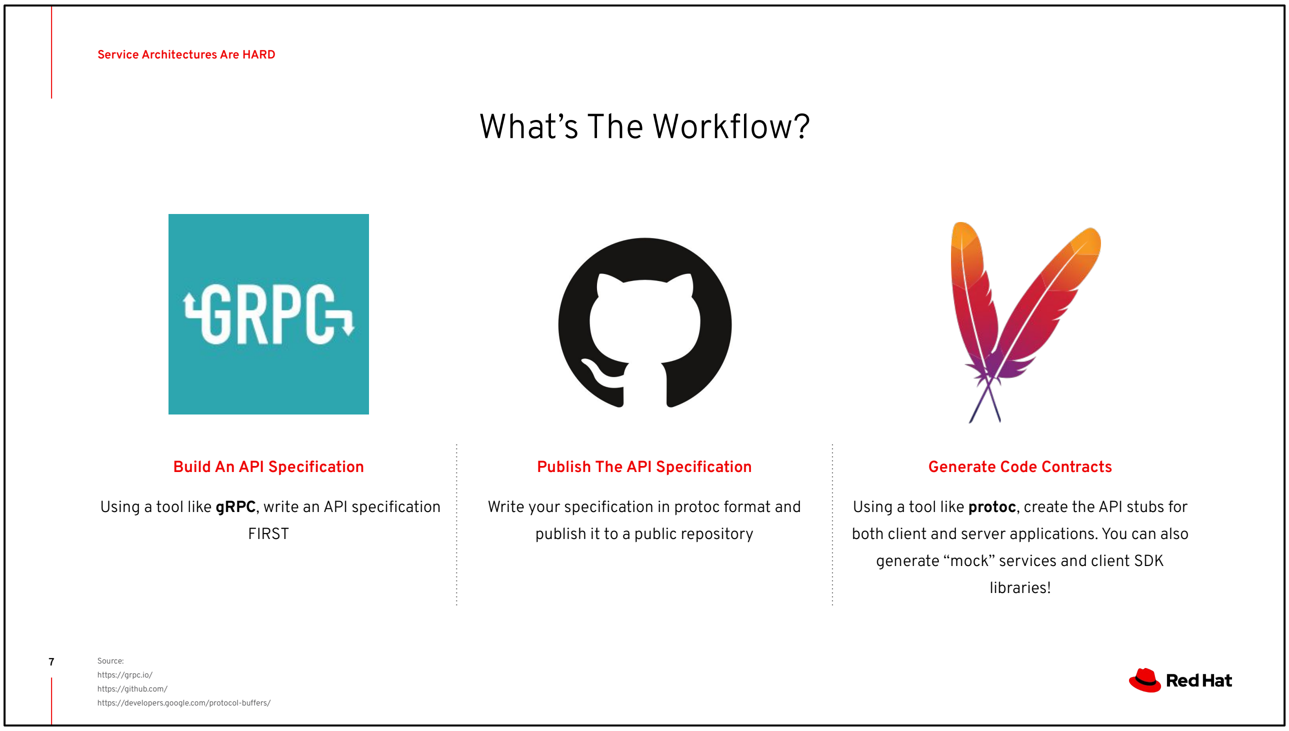
Source:
https://swagger.io/docs/specification/about/
https://www.apicur.io/
https://openapi-generator.tech/

There are a number of "Best Practices" around developing distributed services. Things like Versioned APIs, circuit breakers, etc… Those are pretty well known, but what seems to be less common is using a Contract-First API development approach.

The first step is to use OpenAPI to create a specification for our API. OpenAPI is based on Swagger, which is a tool originally designed to create API documentation from living code. It has since evolved to work either direction. You can write your OpenAPI specification and then generate code from it, OR you can write your service code and generate OpenAPI specifications from it.

We then publish our API specification to somewhere that our teammates, other teams, customers, or partners can access the specification. Once you have the specification, you can start coding against it because you KNOW what the API will look like. It's a contract which the service has promised to fulfill.

From that contract we can generate implementation or stub code for our services. This means that a front-end app consuming our API can have their JavaScript or Typescript code created AUTOMATICALLY!!! Some frameworks and toolkits are even already OpenAPI aware, so you can just tell your application the location of your OpenAPI contract and the REST endpoints will be wired up automatically!



What's The Workflow?

Build An API Specification

Using a tool like **gRPC**, write an API specification FIRST

Publish The API Specification

Write your specification in protoc format and publish it to a public repository

Generate Code Contracts

Using a tool like **protoc**, create the API stubs for both client and server applications. You can also generate "mock" services and client SDK libraries!

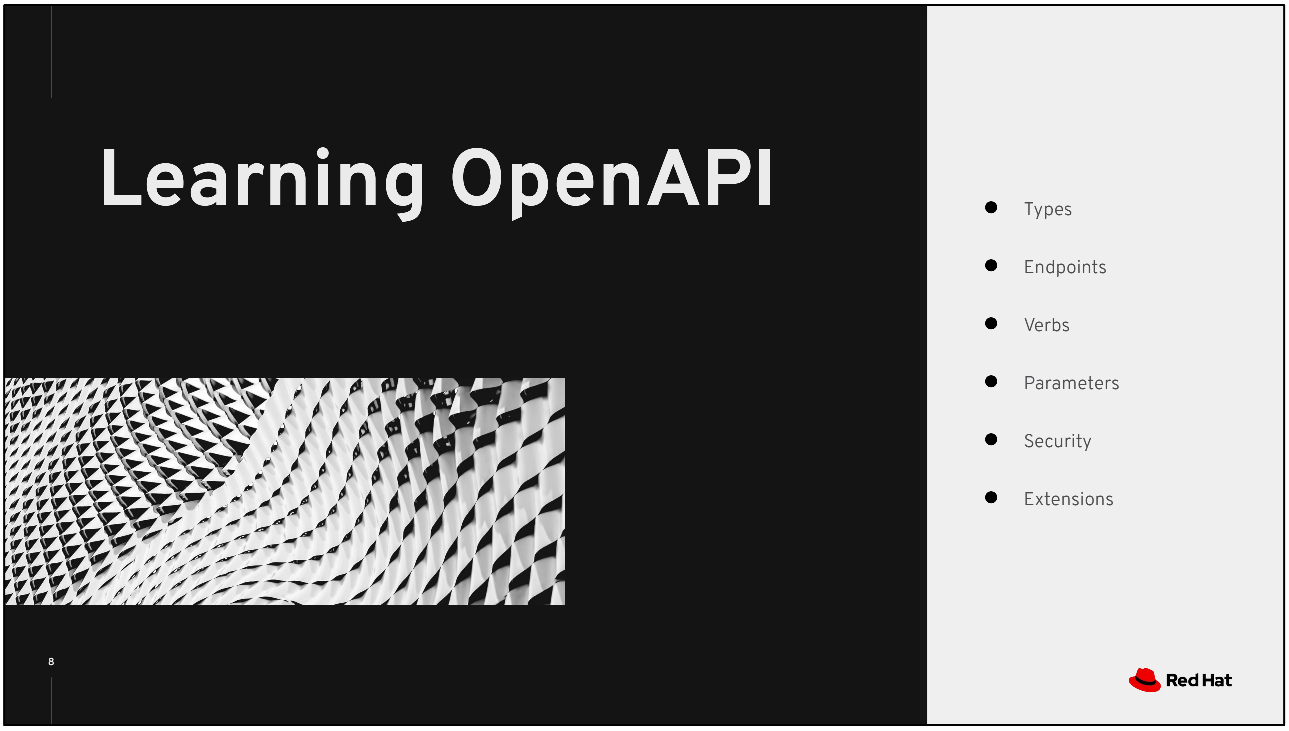
Source:
https://grpc.io/
https://github.com/
https://developers.google.com/protocol-buffers/

Another valid option for doing contract-first API development is to use gRPC:

gRPC is a modern open source high performance RPC framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication. It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services.

gRPC specifications are written in a format called "protocol buffers" which is it's own, small, domain-specific language.

There are other solutions for contract-first, but OpenAPI and gRPC seem to be the most widely adopted. Others include: RAML, RSDL, OData, APIBlueprint, I/O Docs, Apache Avro, etc.…

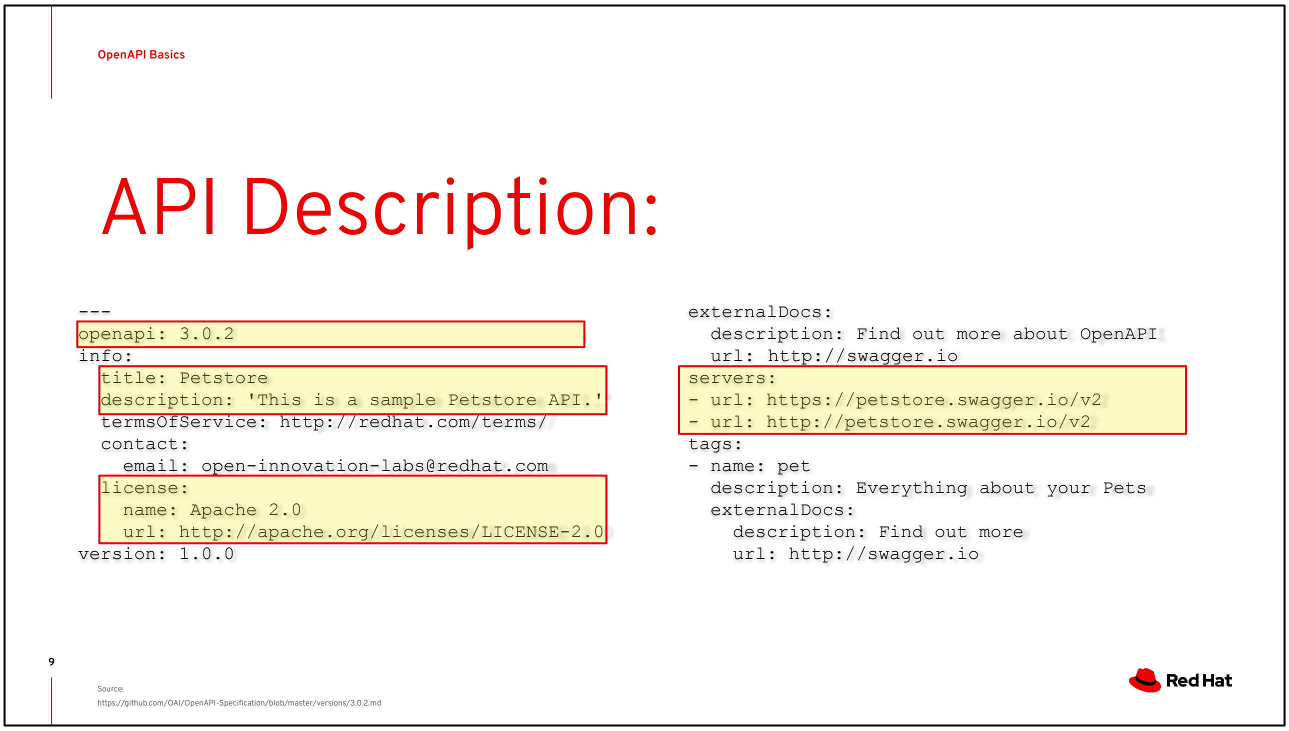


Learning OpenAPI

- Types
- Endpoints
- Verbs
- Parameters
- Security
- Extensions

Red Hat

Because of it’s massive amount of available tooling, we’re going to focus on OpenAPI for now. There is significant documentation for gRPC available on-line if you should choose to go that direction as well.



API Description:

```yaml
---
openapi: 3.0.2
info:
  title: Petstore
  description: 'This is a sample Petstore API.'
  termsOfService: http://redhat.com/terms/
  contact:
    email: open-innovation-labs@redhat.com
  license:
    name: Apache 2.0
    url: http://apache.org/licenses/LICENSE-2.0
version: 1.0.0
```

```yaml
externalDocs:
  description: Find out more about OpenAPI
  url: http://swagger.io
servers:
- url: https://petstore.swagger.io/v2
- url: http://petstore.swagger.io/v2
tags:
- name: pet
  description: Everything about your Pets
  externalDocs:
    description: Find out more
    url: http://swagger.io
```

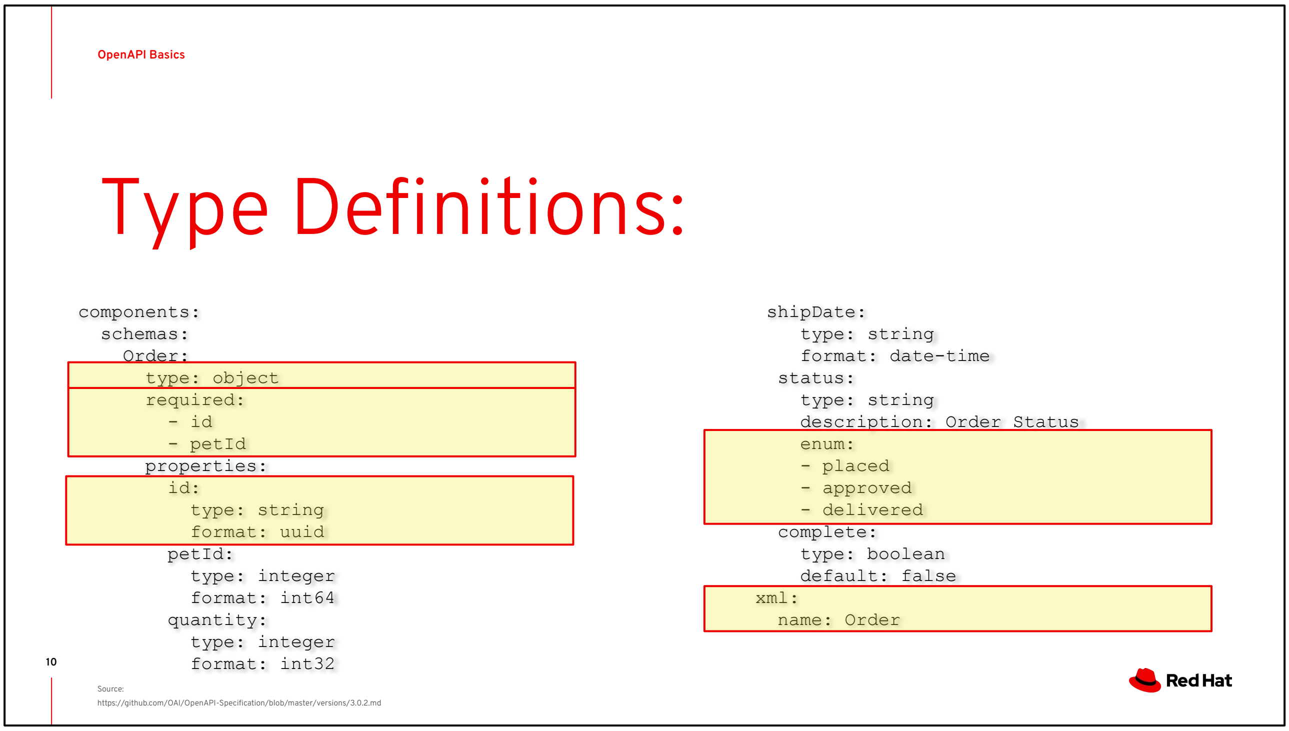
Source:
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md

Red Hat

OpenAPI documents are written in either YAML or JSON.

- They start with a declaration of which version of the OpenAPI specification we are going to use to describe our API.
- The title and description are used when generating client SDKs from code generators
- The “servers” section can be used to set up appropriate CORS and XSS configurations
- The license lets your customers and partners know how they may re-use this content.

Everything else here is informational and pretty much optional, though still good practice for us to include.



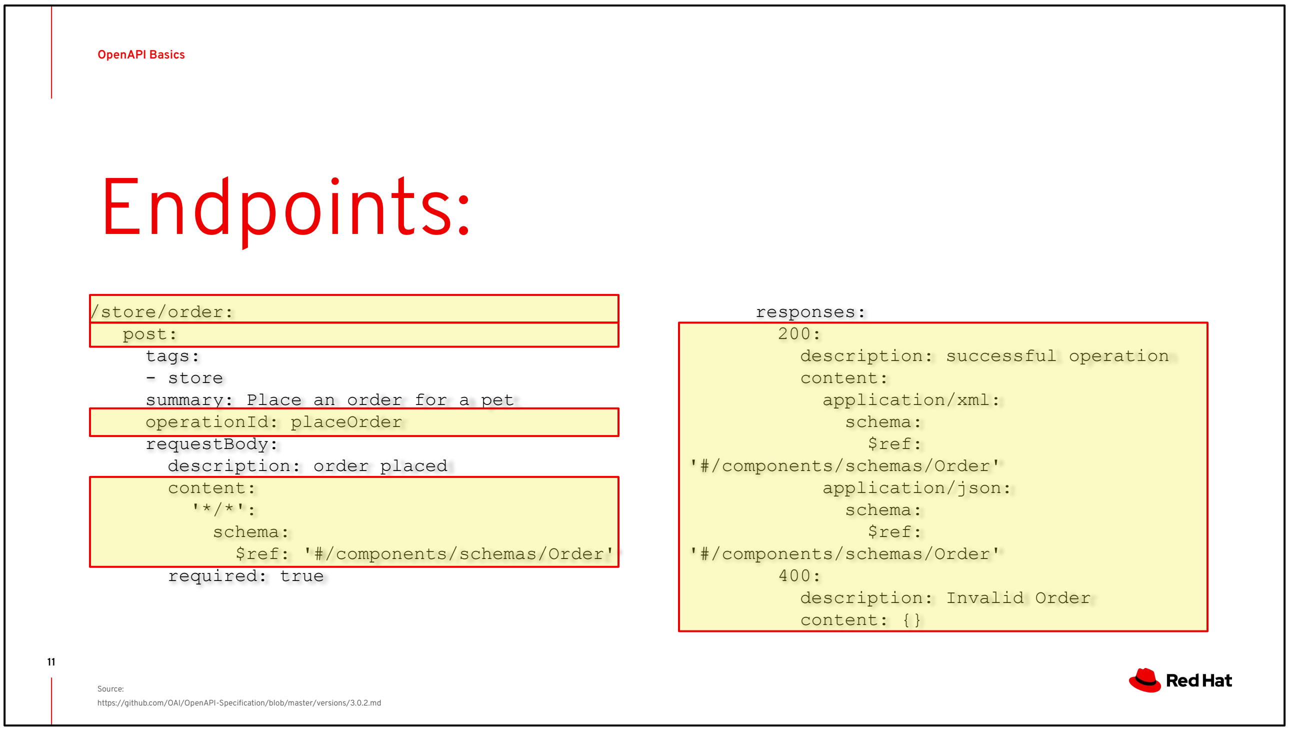
Type Definitions:

```
components:
  schemas:
    Order:
      type: object
      required:
        - id
        - petId
      properties:
        id:
          type: string
          format: uuid
        petId:
          type: integer
          format: int64
        quantity:
          type: integer
          format: int32
        shipDate:
          type: string
          format: date-time
        status:
          type: string
          description: Order Status
          enum:
          - placed
          - approved
          - delivered
        complete:
          type: boolean
          default: false
      xml:
        name: Order
```

Source:
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md

In a section called "components" and a subsection called "schemas", we can define the data types which will be produced and consumed by our API. For example, here we see an "Order" object defined. These schemas are defining the resources our API will be producing/consuming.

- When defining a new type, you specify the type as "object"
- Next, you can define which fields are required to be valid
- Each field can be a primitive type or a reference to another object type, also format
- If you wish to support XML, you can define what the top-level XML element will be named
- Enumerated types are also supported



Endpoints:

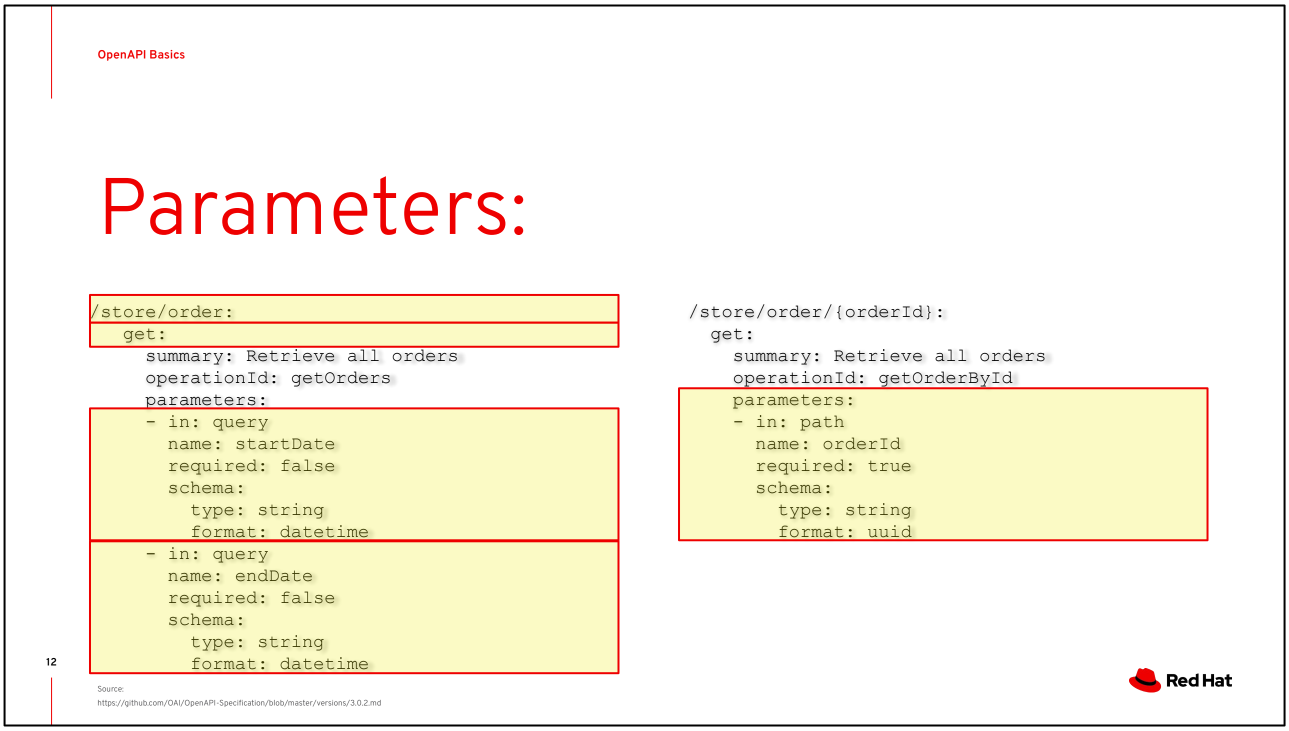
```
/store/order:
  post:
    tags:
    - store
    summary: Place an order for a pet
    operationId: placeOrder
    requestBody:
      description: order placed
      content:
        '*/*':
          schema:
            $ref: '#/components/schemas/Order'
      required: true
```

```
      responses:
        200:
          description: successful operation
          content:
            application/xml:
              schema:
                $ref:
'#/components/schemas/Order'
            application/json:
              schema:
                $ref:
'#/components/schemas/Order'
        400:
          description: Invalid Order
          content: {}
```

Source:
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md

Here we have defined a REST endpoint which handles a POST verb. The content of the POST body is defined as accepting the "Order" type we showed in the previous slide. We have also defined that our API consume and produces both JSON and XML.

- The PATH for the endpoint
- The HTTP Verb for this operation
- The "operationId" is used by many code-generators to create method names for both the server and client SDK implementations
- The content of the request is described either directly or using a reference to a previously defined type
- Response codes and their associated bodies are defined either directly or using a reference to a previously defined type



Parameters:

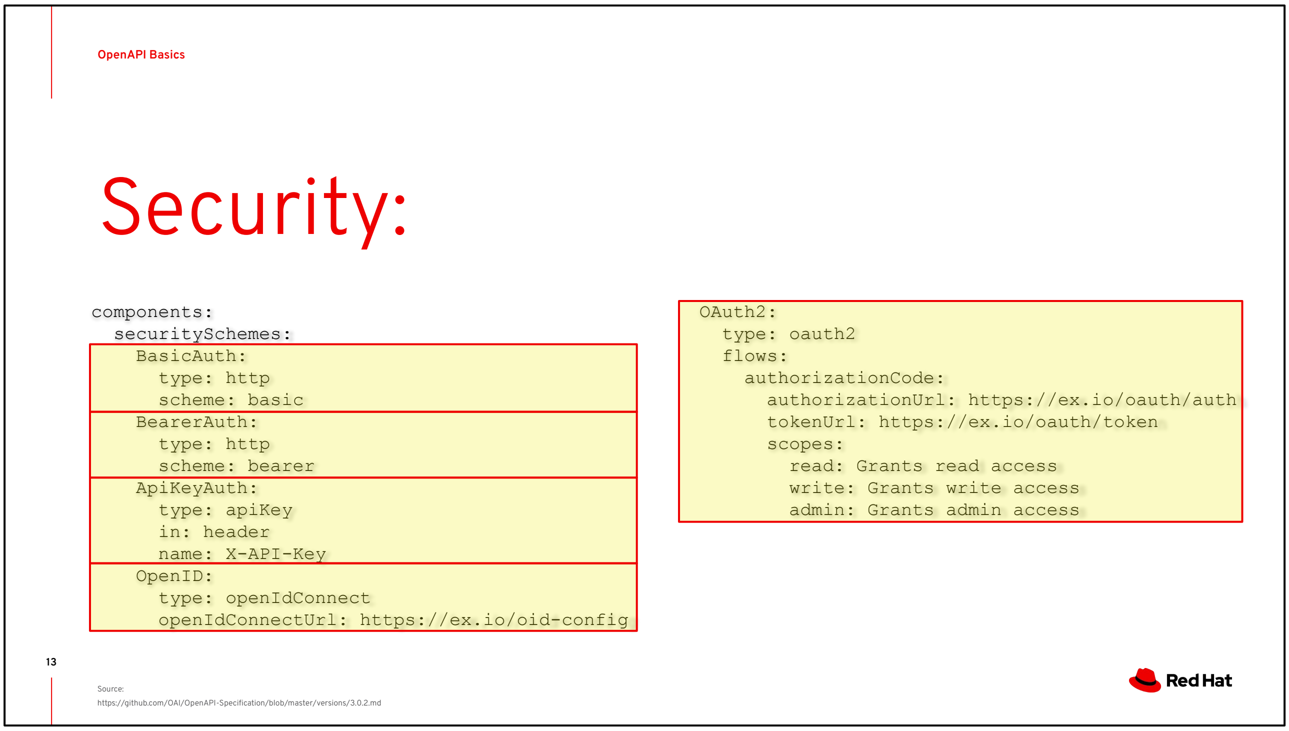
```
/store/order:
  get:
    summary: Retrieve all orders
    operationId: getOrders
    parameters:
    - in: query
      name: startDate
      required: false
      schema:
        type: string
        format: datetime
    - in: query
      name: endDate
      required: false
      schema:
        type: string
        format: datetime
```

```
/store/order/{orderId}:
  get:
    summary: Retrieve all orders
    operationId: getOrderById
    parameters:
    - in: path
      name: orderId
      required: true
      schema:
        type: string
        format: uuid
```

Source:
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md

Red Hat

We can also specify the parameters which can be passed for various endpoints in our API. Here we see query parameters and path parameters defined and we can also see that some are required while others are not.



Security:

```
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
    BearerAuth:
      type: http
      scheme: bearer
    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key
    OpenID:
      type: openIdConnect
      openIdConnectUrl: https://ex.io/oid-config
```

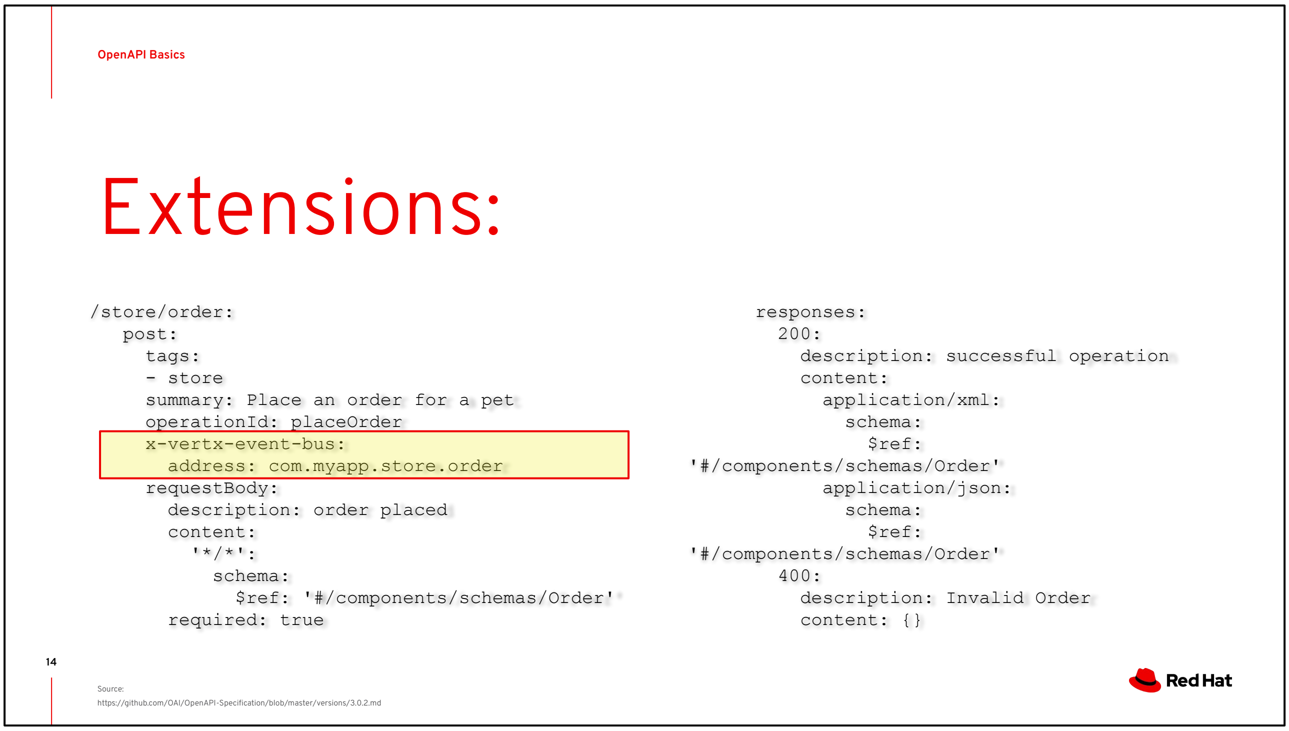
```
OAuth2:
  type: oauth2
  flows:
    authorizationCode:
      authorizationUrl: https://ex.io/oauth/auth
      tokenUrl: https://ex.io/oauth/token
      scopes:
        read: Grants read access
        write: Grants write access
        admin: Grants admin access
```

Source:
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md

Red Hat

Here we have defined a REST endpoint which handles a POST verb. The content of the POST body is defined as accepting the “Order” type we showed in the previous slide. We have also defined that our API consume and produces both JSON and XML.

- The PATH for the endpoint
- The HTTP Verb for this operation
- The “operationId” is used by many code-generators to create method names for both the server and client SDK implementations
- The content of the request is described either directly or using a reference to a previously defined type
- Response codes and their associated bodies are defined either directly or using a reference to a previously defined type



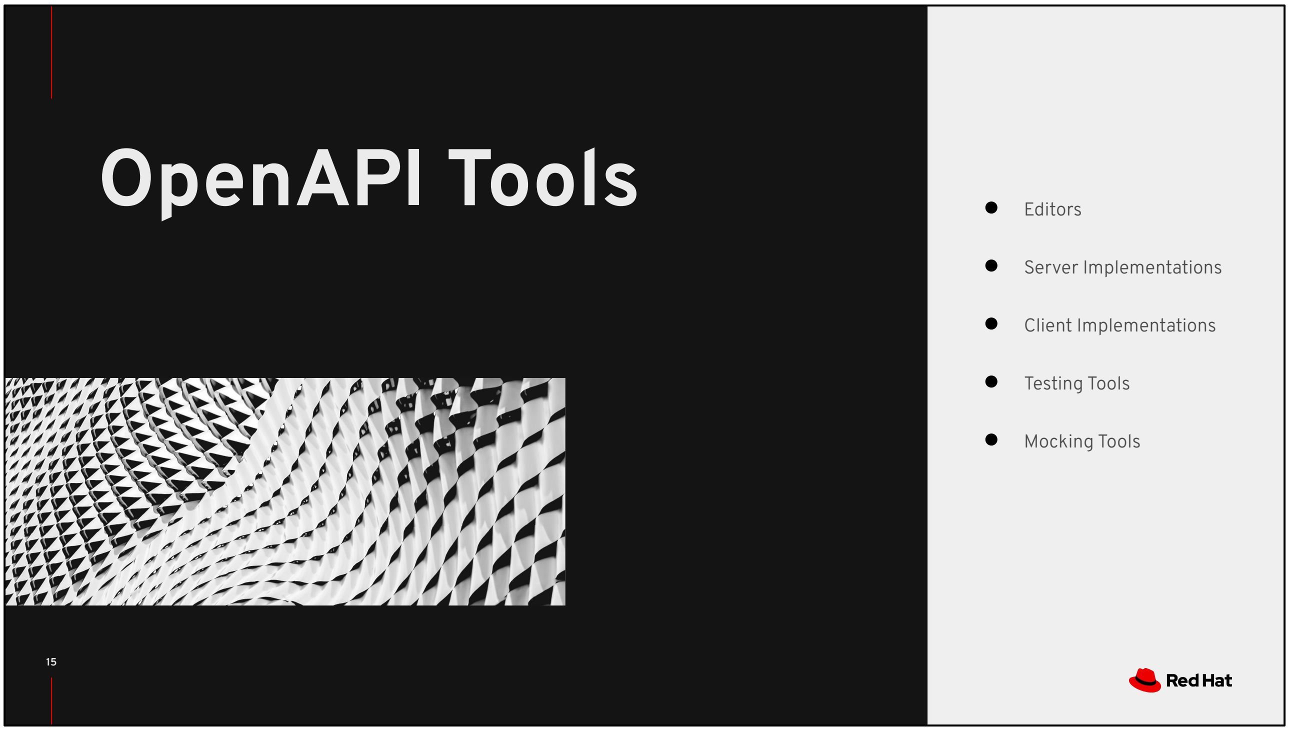
Extensions:

```
/store/order:
   post:
     tags:
     - store
     summary: Place an order for a pet
     operationId: placeOrder
     x-vertx-event-bus:
       address: com.myapp.store.order
     requestBody:
       description: order placed
       content:
         '*/*':
           schema:
             $ref: '#/components/schemas/Order'
       required: true
```

```
      responses:
        200:
          description: successful operation
          content:
            application/xml:
              schema:
                $ref:
'#/components/schemas/Order'
            application/json:
              schema:
                $ref:
'#/components/schemas/Order'
        400:
          description: Invalid Order
          content: {}
```

Source:
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md

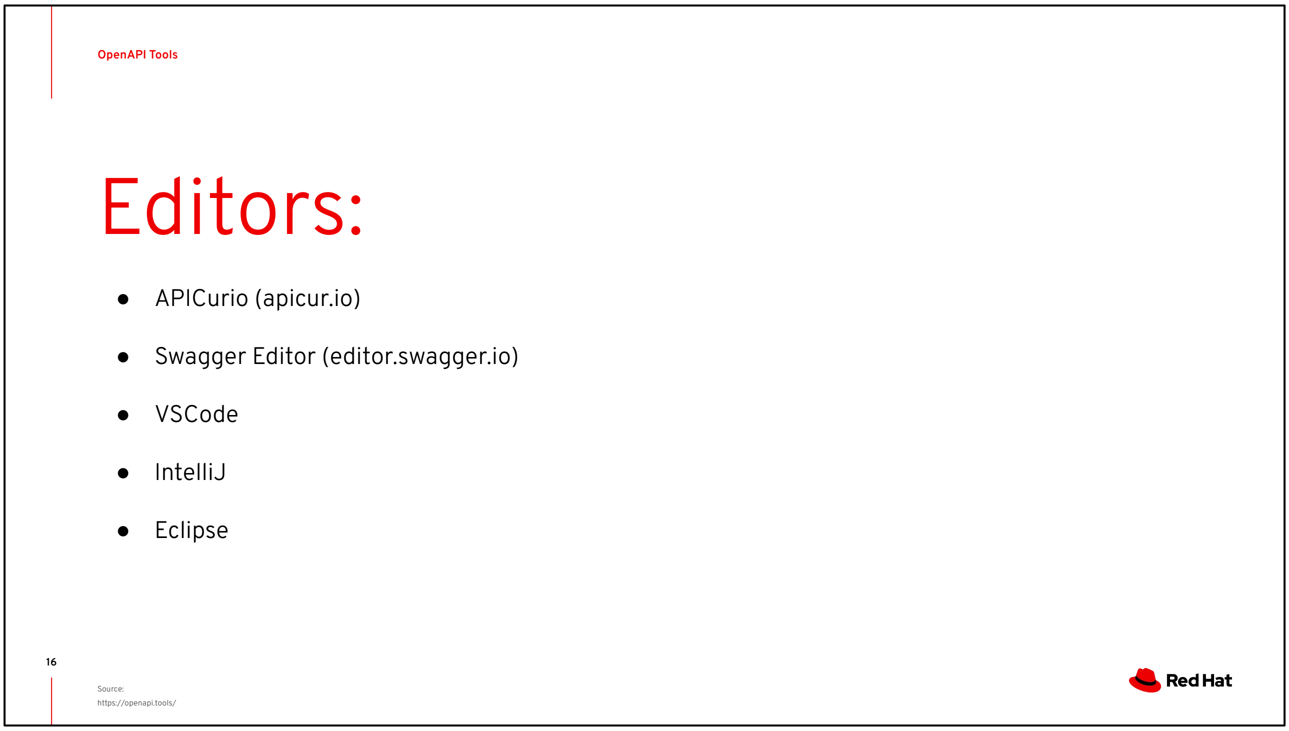
The OpenAPI specification leaves room for extension, so if you want to implement something specific to a toolkit/framework/platform; you can. For example, here we see a field which start with "x-", indicating it's an extension to the OpenAPI specification standard. This one is specific to writing applications using Vert.x and allows us to tell Vert.x more information about our API in order to help it integrate with the OpenAPI spec in a way which is toolkit specific. Sometimes you will see extensions for database schema generation, or security support systems, etc… The OpenAPI spec can be extended without writing any custom code, but you may like to extend OpenAPI Generator so that it can generate more/better code for your organization and even further reduce the amount of work required.



OpenAPI Tools

- Editors
- Server Implementations
- Client Implementations
- Testing Tools
- Mocking Tools

There are many tools around OpenAPI which make it possible to do faster and more asynchronous development so that teams, developers, customers, and partners can develop clients and servers reliably.



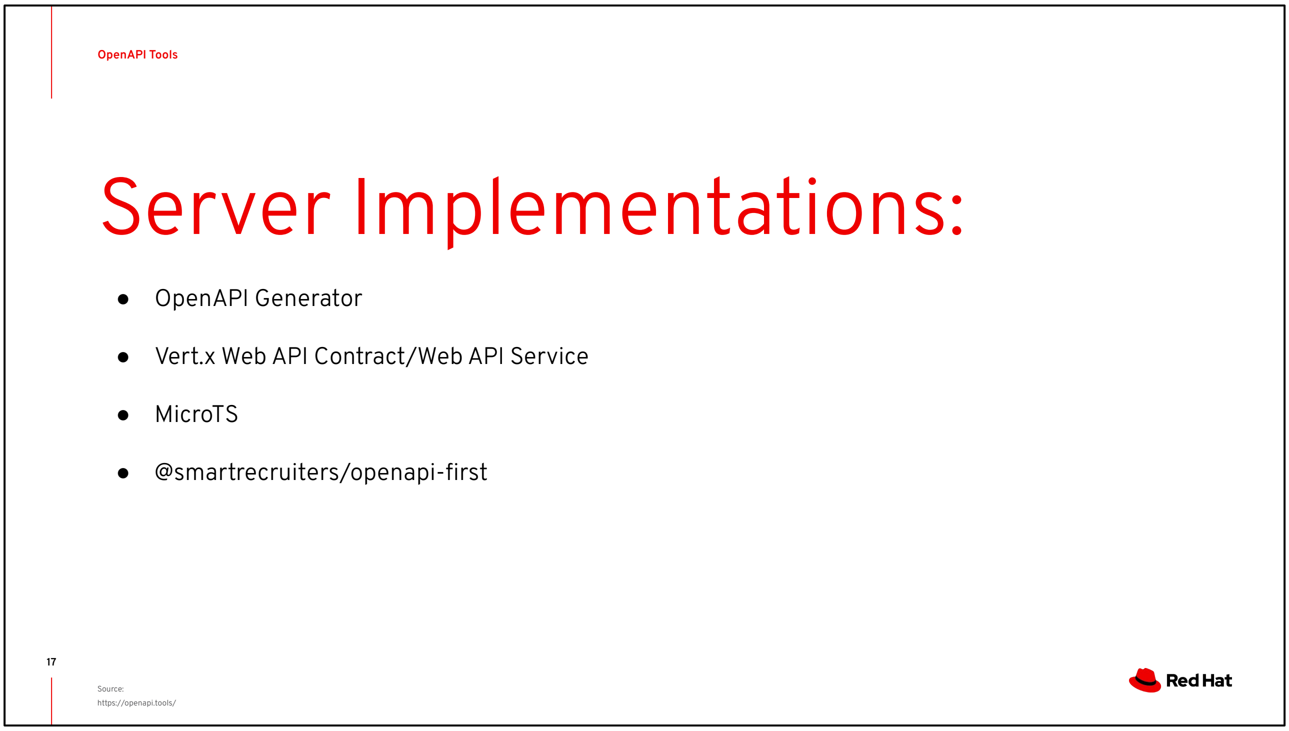
Editors:

- APICurio (apicur.io)
- Swagger Editor (editor.swagger.io)
- VSCode
- IntelliJ
- Eclipse

Source:
https://openapi.tools/

Red Hat

There are a number of editors and tools which can be used to create/edit/validate an OpenAPI specification. Red Hat actually has people working on Apicur.io, but Swagger Editor and IDE plugins are also viable options.



Server Implementations:

- OpenAPI Generator
- Vert.x Web API Contract/Web API Service
- MicroTS
- @smartrecruiters/openapi-first

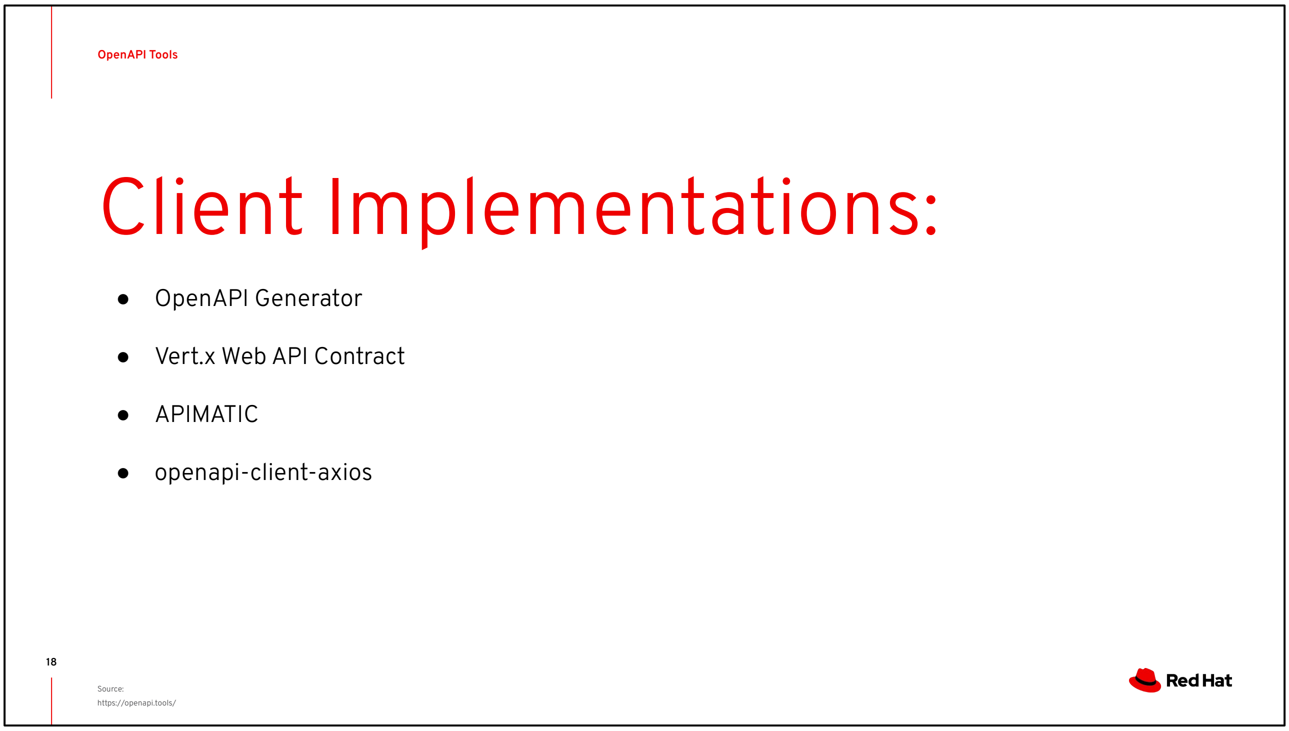
Source:
https://openapi.tools/

OpenAPI Generator can GENERATE code for a number of different platforms (Java, JavaScript, Go, Erlang, Elixir, Typescript, Ruby, Python, etc…) and frameworks (JAX-RS, Spring, Express, Phoenix, etc…)

Vert.x toolkit includes the ability to generate RESTful endpoints and even generate code for Service stubs

MicroTS generates a NodeJS/Express service using TypeScript

openapi-first also generates a NodeJS/Express server, but much of the boilerplate code is generated on-the-fly at runtime and we just write business logic.



Client Implementations:

- OpenAPI Generator
- Vert.x Web API Contract
- APIMATIC
- openapi-client-axios

Source:
https://openapi.tools/

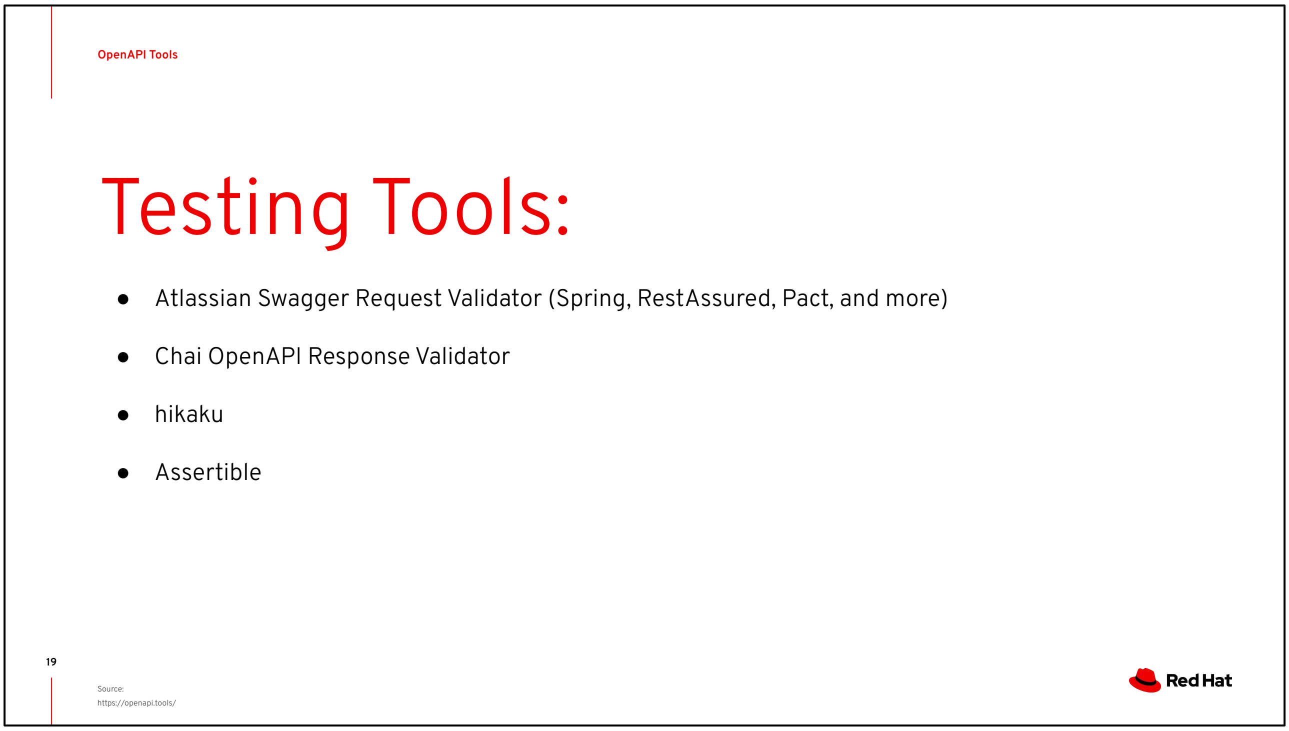
Red Hat

OpenAPI Generator can GENERATE code for a number of different platforms (Java, JavaScript, Go, Erlang, Elixir, Typescript, Ruby, Python, etc…) and frameworks (JAX-RS, Spring, Express, Phoenix, etc…)

Vert.x toolkit includes the ability to generate client implementations in ANY of the supported Vert.x languages (Java, JavaScript, Typescript, Clojure, Scala, Kotlin, Groovy, etc..)

APIMatic is a SaaS solution for generating an SDK package in any number of languages based on an OpenAPI Spec

openapi-client-axios can use an OpenAPI spec to create a fully async-enabled Axios client for your JavaScript and Typescript applications



Testing Tools:

- Atlassian Swagger Request Validator (Spring, RestAssured, Pact, and more)
- Chai OpenAPI Response Validator
- hikaku
- Assertible

Source:
https://openapi.tools/

Red Hat

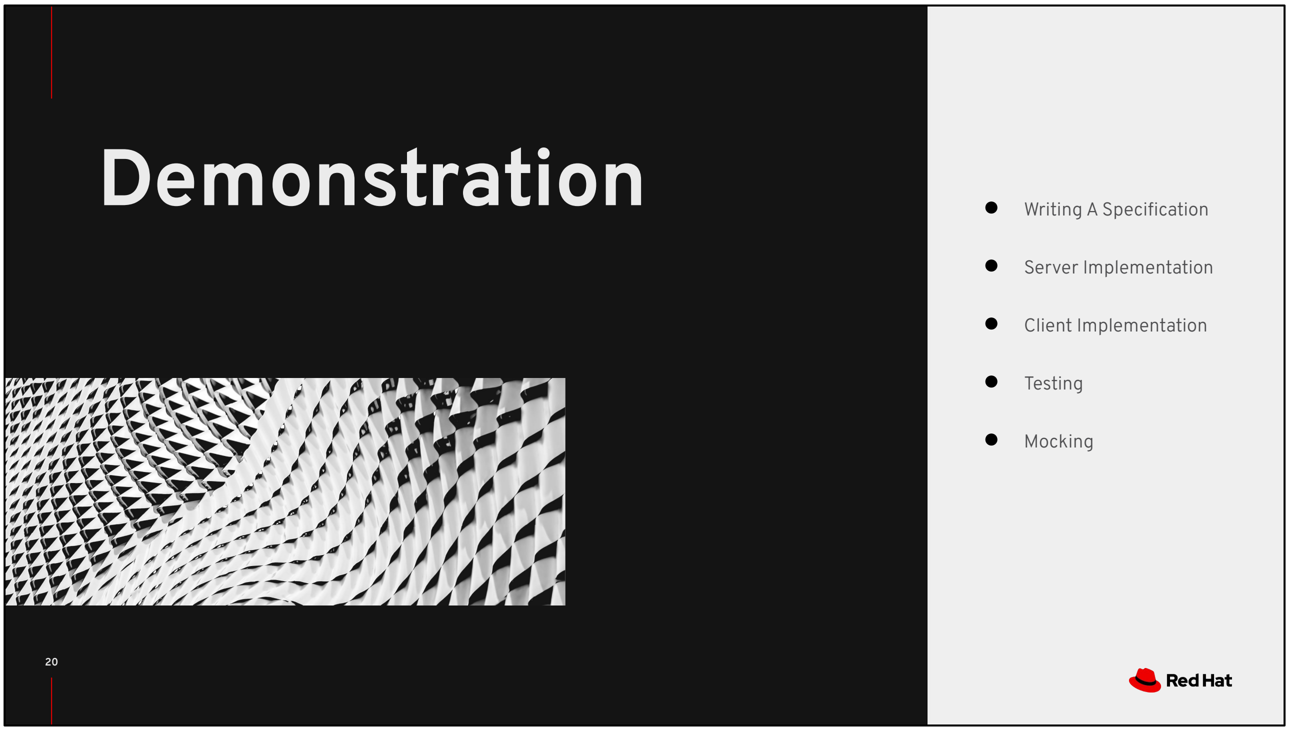
Dredd - Language-agnostic command-line tool for validating API description document against backend implementation of the API

Atlassian Swagger Request Validator - A series of libraries for several Java frameworks and toolkits which adds API contract validation to your tests/applications

Chai OpenAPI Response Validator - Simple Chai support for asserting that HTTP responses satisfy an OpenAPI spec.

hikaku - A library that tests if the implementation of a REST-API meets its specification.

Assertible - Import an OpenAPI specification into Assertible to generate tests that validate JSONSchema responses and status codes on every endpoint.

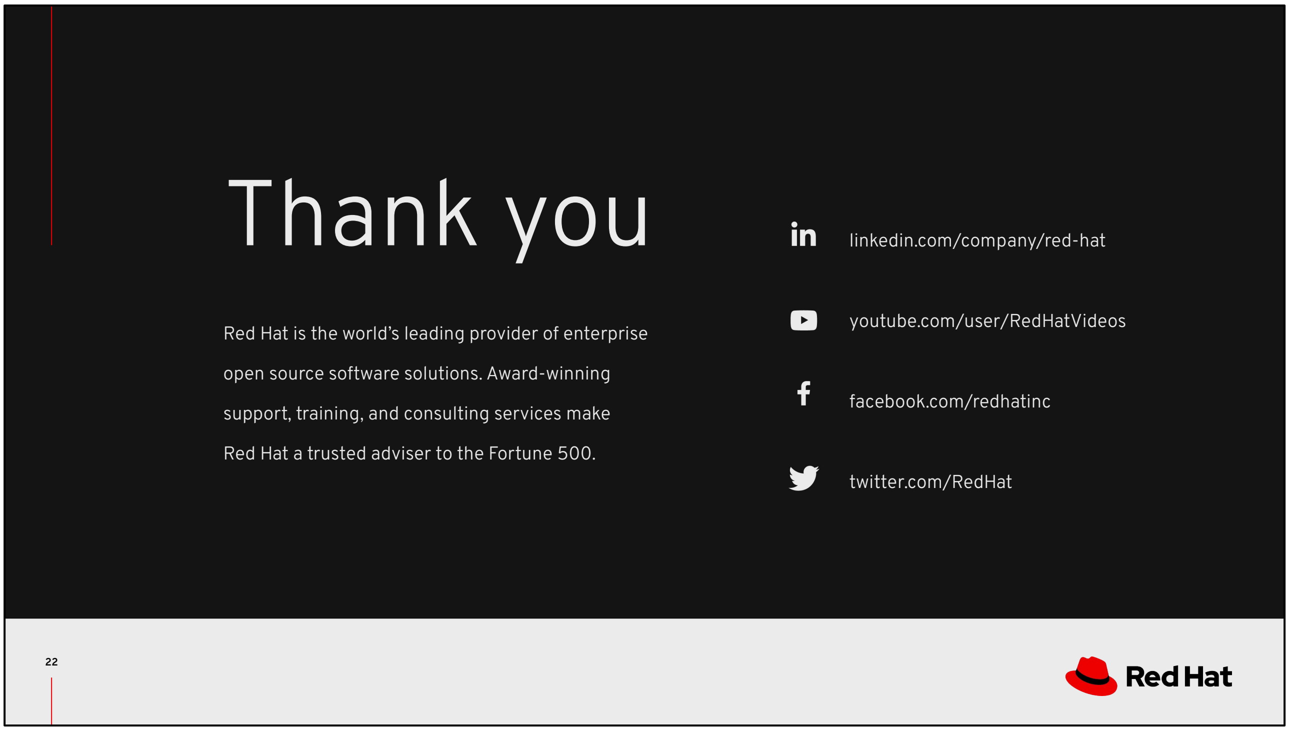


Demonstration

- Writing A Specification
- Server Implementation
- Client Implementation
- Testing
- Mocking

So here’s where the rubber meets the road.

https://red.ht/3SWVuB0



Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat

If you have more questions about cloud-native application development, I will be around all day and happy to answer questions. Also, at the Red Hat Integration table you can talk to Hugo Guerroro and he should be able to assist you as well!!!