Microservices in Java: Server Side

Idaho State University | Computer Science

Isaac Griffith

CS 3321
Department of Computer Science
Idaho State University

ROAR

Netwoking Basics

IDG

CS 3321

ROAR

Servers and Services

- **Server** - an overloaded term which may refer to either of the following:
  - A dedicated computer connected to a network which provides files or services
  - A program running on a computer which provides one or more services and to which we connect via a port using a socket
- **Service** - An application that provides data storage, manipulation, presentation, communication, or other capability
  - typically implemented using a client-server or peer-to-peer architecture and application layer network protocols (e.g., http, ftp, or snmp)

URIs

- **URI** - Uniform Resource Identifier
  - **URL** is a subtype of this for internet resources such as webpages
  - But, the concept is much more general than that
- URI Syntax
  - `http(s)://<address>[:port][/resource]*`

Idaho State University
Computer Science

IP Addresses

- Every machine connected to a network/the internet must be able to be identified
- This ensures that information can be routed correctly
- Thus, we came up with the idea of IP addresses
  - A theoretically unique number for every machine connected to the network
  - The most common version of IP is version 4
  - Unfortunately, with the advent of smartphones, IoT devices, etc. we ran of out address 11/25/2019
  - But, smart engineers managed to work around that, but something else was needed, thus we have IPv6

ROAR

Idaho State University
Computer Science

IP addresses

- Even with IPv6, IPv4 addresses are still more commonly known
- They are 4 numbers ranging between 0 and 255 separated by dots
- For example:
  - 192.168.0.1
  - 134.50.105.172
  - 127.0.0.1 (special loopback device)
- Each IP Address is related to the underlying network device on your machine
- Each network device has its own unique address called the MAC (Media Access Control) address

ROAR

Idaho State University
Computer Science

IP Addresses

- You can find your own IP address using one of the following tools on the command line:
  - **ifconfig** - linux/mac
  - **ipconfig** - windows
- Or through your network settings if you want to use the windowed approach

ROAR

Ports

- IP addresses are used to ensure that information is routed correctly
- But, we also need to be able to send and receive specific types of information.
- Ports allow us to setup multiple programs (servers/services) to receive information
- Ports act like digital/logical mailboxes to which different packets of information can be delivered.
- Thus, a single machine can specify what types of information it is listening for and what protocols it uses

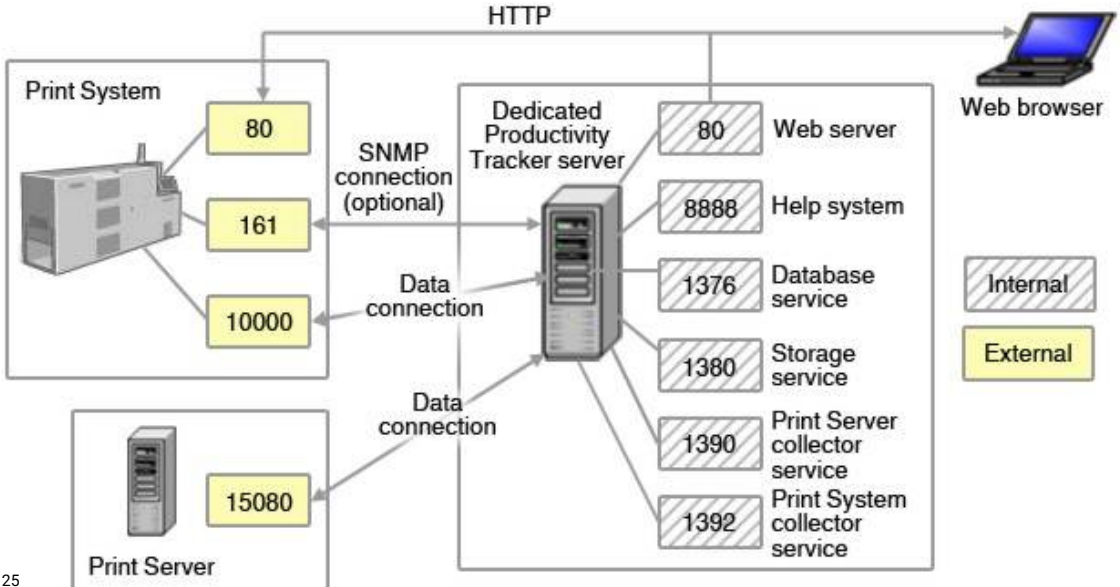
Ports

- Common ports that are open on many servers/computers include
  - **20,21** - FTP
  - **22** - ssh
  - **80,8080** - http/web (default)
  - **3306** - MySQL
- The well-known parts are in the range 0 - 1023
- The range of registered ports is 1024 - 49151
- The range of dynamic, private, or ephemeral ports is 49152 - 65535

Ports

- Thus when setting up a new service, it tends to be wise to make sure you are not using a well registered port.
  - But, we are using HTTP, so port 80 can be used (if not already in use by another program, may want to consider NGinX to get around this)
- You can determine what ports are used on your system with tools such as:
  - nmap
  - netstat

Ports


HTTP
Print System
80
161
10000
SNMP connection (optional)
Data connection
Data connection
15080
Print Server
Dedicated Productivity Tracker server
80
Web server
8888
Help system
1376
Database service
1380
Storage service
1390
Print Server collector service
1392
Print System collector service
Web browser
Internal
External

Idaho State University
Computer Science

REST?

What is REST?

- **REST** - Representational State Transfer
- RESTful systems should
  - Client-server architecture
  - Operations should be stateless (i.e., do not remember between calls)
    - No information is retained by the server between sessions
  - use a layered approaches
  - support code on demand
  - have a uniform interface
- **Goal:** systems with high performance, scalability, modifiability, visibility, portability, and reliability

ROAR

HTTP Methods

- The five basic/useful methods for our work are:
  - **GET** - requests the target resource to transfer a representation of its current state. Only retrieves data and should have no other effect.
  - **POST** - requests that the target process the representation enclosed in the request
  - **PUT** - requests that the target resource create or update its state with the state defined by the representation enclosed in the request
  - **DELETE** - request that the target resource delete its state
  - **PATCH** - requests that the target resource modify its state according to the partial updated defined in the representation enclosed in the request.

Testing HTTP Methods

- HTTP methods, when combined with a URI effectively construct a RESTful API
- For example, GitHub (and many other sites) has a RESTful API
  - Example: `curl https://api.github.com/search/users?q="grifisaa"`
  - This returns a JSON document containing information about my personal github profile
- You can use curl to transfer a URI using any of the specific HTTP methods, as long as you can access the API
  - Note, you can also provide curl with authentication headers as well, but that is beyond the scope

Javalin Microservices

IDG

CS 3321

ROAR

Idaho State University
Computer Science

Hello World

```java
import io.javalin.Javalin;

public class HelloWorld {
  public static void main(String[] args) {
    Javalin app = Javalin.create().start(7000);
    app.get("/", ctx -> ctx.result("<h1>Hello World!</h1>"))
  }
}
```

1. This creates a new service on the localhost running on port 7000
2. When we send a **GET** request to the route “/”, it returns `<h1>Hello World</h1>`
   - “/” is the root route
   - the full url for this route would be - http://localhost:7000/

Let’s see this in action

ROAR

Idaho State University
Computer Science

How do we start using Javalin?

- First, you probably should use something like Gradle or Maven to manage your dependencies
- I prefer Gradle, so I added the following to my `build.gradle` file

```
dependencies {
  implementation 'io.javalin:javalin:4.0.1'
}
```

- Additionally, you will need add the following dependencies (for logging and json processing)

```
dependencies {
  implementation 'org.slf4j:slf4j-simple:1.7.31'
  implementation 'com.fasterxml.jackson.core:jackson-databind:2.12.4'
}
```

ROAR

Handlers

- To process a “route” we need a handler for that particular route, and we simply add it to our server
- There are five types of handlers that can be used, but the three main ones are:
  - *before-handlers* - matched before every request
  - *endpoint-handlers* - these define the api (the get handler in the prior code)
  - *after-handlers* - run after every request (even is an exception occurs)

Handlers

- Each of the three handlers require three parts
  - A verb (one of: `before`, `get`, `post`, `put`, `patch`, `delete`, `after`)
  - A path (i.e., `/`, `/status`, `/user/{name}`)
  - A handler implementation, which can be defined as follows
    - Using an lambda function: `ctx -> { ... }`, or
    - implementing the interface `io.javalin.http.Handler`

Handlers

- In addition to the basic operations which we can define one at a time.
- Often, we need to implement multiple operations a particular path.
- Thus, we can provide a `CrudHandler` instance for the particular route

Handlers

```
app.routes(() -> {
  crud("users/{user-id}", new UserController())
});
```

- `UserController` would need to implement the `CrudHandler` interface which provides the following methods:
  - `getAll(ctx)`
  - `getOne(ctx, resourceId)`
  - `create(ctx)`
  - `update(ctx, resourceId)`
  - `delete(ctx, resourceId)`
- Where `resourceId` happens to be the parameter `{user-id}` in this case

Path Parameters

- We can extract data provided in path-parameters using:
  - `ctx.pathParam("key")`
- Example:

```
app.get("/hello/{name}", ctx -> {
  ctx.result("Hello: " + ctx.pathParam("name"));
})
app.get("/hello/<name>", ctx -> {
  ctx.result("Hello: " + ctx.pathParam("name"));
})
```

Idaho State University
Computer Science

JSON Data

- The typical data format to send between the client and server is JSON
- For our **GET** routes, we can simply provide the JSON equivalent of an object by calling
  - `ctx.json(obj)`
- For our **POST** routes, which receive data, we can do the following
  1. validate that the content type is json: `ctx.contentType().equals("application/json")`
  2. extract the json data into the expected class: `obj = ctx.bodyAsClass(Class.class)`
- You can test the post with curl as follows:
  - `curl -X POST http://localhost:7000/api/user -H 'Content-Type: application/json' -d '{"firstName":"Foo","lastName":"Bar","email":"foobar@isu.edu"}'`

ROAR

Idaho State University
Computer Science

Robust Server

```
public static void main(String[] args) {
  QueuedThreadPool queuedThreadPool =
              new QueuedThreadPool(200, 8, 60000);

  Javalin app = Javalin.create(config -> {
    config.server(() -> {
      Server server = new Server(queuedThreadPool);
      return server;
    }
  }).start(7000)

  app.routes(() -> {
    get("/hello", ctx -> ctx.result("Hello"));
  });
}
```


ROAR

Are there any questions?