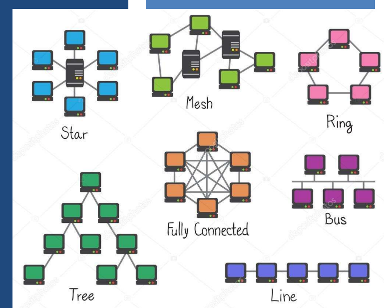
TCSS 558: APPLIED DISTRIBUTED COMPUTING



Distributed Systems: Ch. 2 Architectures

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

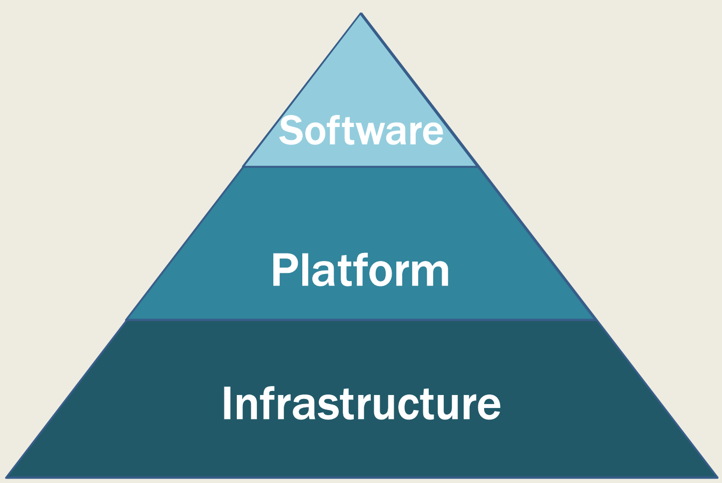
OBJECTIVES

- Feedback from 10/5
- Ch. 2 - Architectural styles
  - Layered
  - Object-based
  - Resource-centered
  - Event-based
- Next:
  Middleware organization

FEEDBACK – 10/5

- IaaS vs. PaaS vs. FaaS
- "What are these really?"

CLOUD COMPUTING STACK



FEEDBACK – 10/5

- IaaS vs. PaaS vs. FaaS
- "What are these really?"
- Each offers a particular type of computing resource as-a-service
- Service means that resources are available to end users via a programmable interface
- Not only can users create, view, update, delete resources via a GUI, but also programmatically
- Why do we want services to be accessible programmatically ?

IAAS / PAAS / FAAS

- Install AWS CLI (Page 15, assignment 0)

```
$sudo apt update
$sudo apt install awscli
```

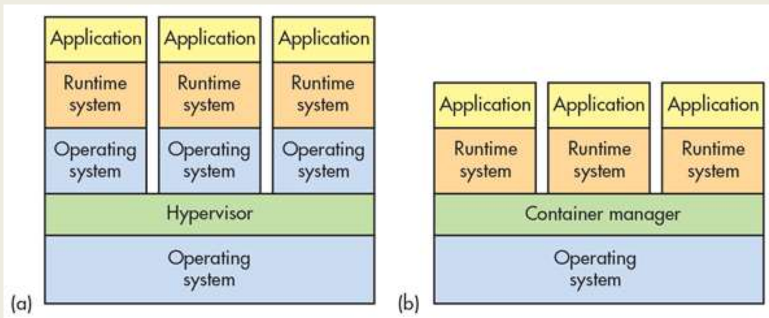
- Configure the AWS CLI with access credentials (pg. 2 & 3)

```
# configure aws cli
$aws configure
```

- Let's inspect these:
- IaaS: aws ec2 help
- PaaS: aws elasticbeanstalk help
- FaaS: aws lambda help

FEEDBACK - 2

- Virtualization and Containerization
  - What is the difference between virtual machines and containers?



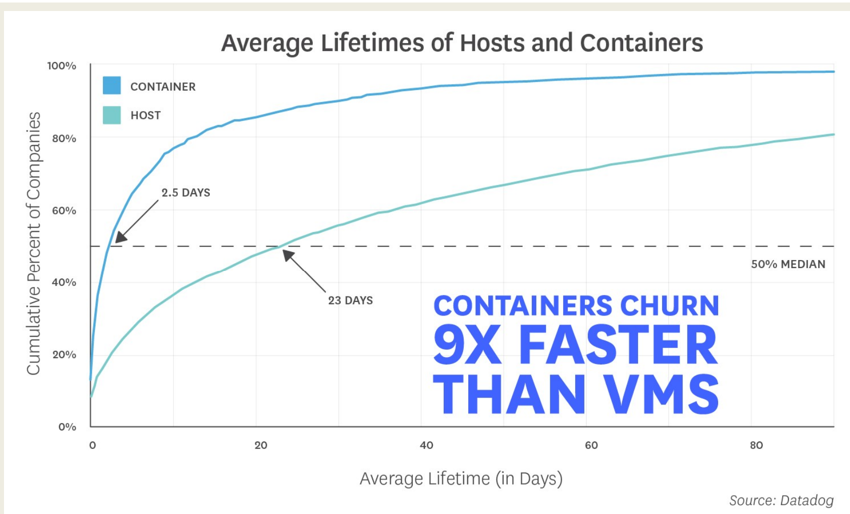
VMs vs. CONTAINERS

Feature	Virtual Machines	Containers
Virtualization	At what level?	
Operating System(s)	Kernels, OSes	
Image Size	KB, MB, GB	
Isolation	At what level?	
Density	How many per machine?	
Boot Time	?	
Memory	Management, allocation, reservation…	
Lifetime (churn)	At what level?	
Overhead	?	

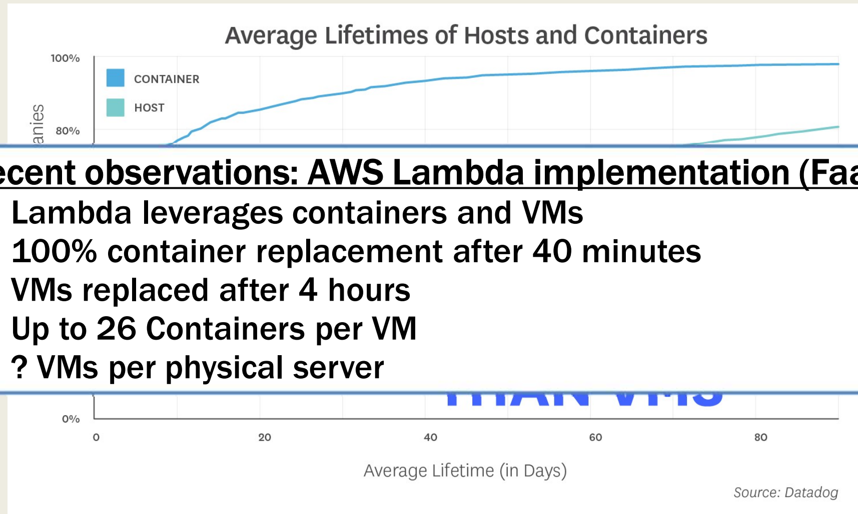
VMs vs. CONTAINERS

Feature	Virtual Machines	Containers
Virtualization	At the HW level	At the OS level
Operating System(s)	Mix different types (kernels): e.g. Ubuntu, Redhat, Windows	Shared kernel just one kernel (uname –a)
Image Size	~1 to 30 GB	10x less: 1 KB to few hundred MB
Isolation	Kernel level	OS Process-level
Density	Dozens per machine	Hundreds per machine
Boot Time	~ a minute	A few seconds
Memory	Reserved memory	Memory unreserved
Lifetime (churn)	Slow churn	Faster churn
Overhead	Higher due to HW abstraction	Lower: little HW abstraction

CHURN OF VMS & CONTAINERS



CHURN OF VMS & CONTAINERS



Recent observations: AWS Lambda implementation (FaaS)

- Lambda leverages containers and VMs
- 100% container replacement after 40 minutes
- VMs replaced after 4 hours
- Up to 26 Containers per VM
- ? VMs per physical server

October 10, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L4.11

FEEDBACK - 3

- Can the lecture slides be available right after class? So far there's a one day delay…

October 10, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L4.12

CH. 2: DISTRIBUTED SYSTEMS ARCHITECTURES

L4.13

DISTRIBUTED SYSTEM ARCHITECTURES

- Logical organization of a distributed system into software components
- Logical: How system is perceived, modeled
  - *The OO/component abstractions*
- Physical – how it really exists
- Middleware
  - Helps separate application from platforms
  - Helps organize distributed components
  - How are the pieces assembled?
  - How do they communicate?
  - How are systems extended? replicated?
  - Provides "realization" of the architecture

October 10, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L4.14

CENTRALIZED VS. DECENTRALIZED DISTRIBUTED SYSTEM ARCHITECTURE

- Tradeoff space: degree of distribution of the system

- Single point-of-failure
- No nodes: vertical scaling
- Always consistent
- Less available (fewer 9s)
- Immediate updates
- No data partitions

- Multiple failure points
- Nodes: horizontal scaling
- Eventually consistent
- More available (more 9s)
- Rolling updates
- Data partitioned or replicated

October 10, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L4.15

ARCHITECTURAL BUILDING BLOCKS

- **Component**: modular unit with well-defined, required, and provided **interfaces** that is replaceable within its environment
- Components can be replaced while system is running
- Interfaces must remain the same
- Preserving interfaces enables interoperability

- **Connector**: enables flow of control and data between components

- Distributed system architectures are conceived using components and connectors

October 10, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L4.16

CH. 2 - ARCHITECTURAL STYLES

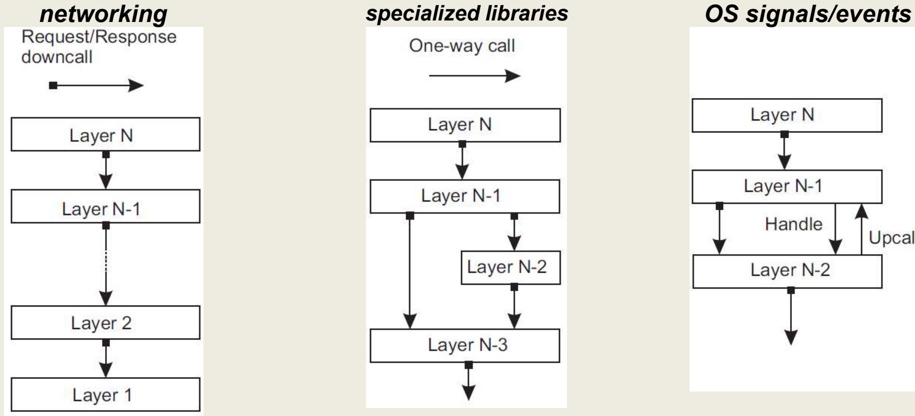
- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

LAYERED ARCHITECTURES

- Components organized in layers
- Component at layer L_j downcalls to lower-level components at layer L_i (where $i < j$)
- Calls go down
- Exceptional cases may produce upcalls

LAYERED ARCHITECTURES - 2

Pure-layered Organization	Mixed-layered organization	Layered w/ upcalls organization
networking	***specialized libraries***	***OS signals/events***

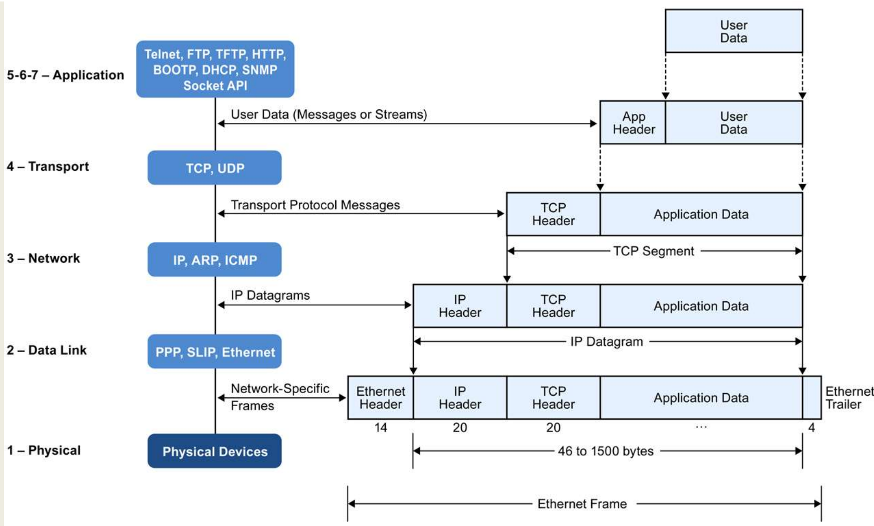


COMMUNICATION-PROTOCOL STACKS

- Example: pure-layered organization
- Each layer offers an interface specifying functions of the layer
- Communication protocol: rules used for nodes to communicate
- Layer provides a **service**
- **Interface** makes service available
- **Protocol** implements communication for a layer

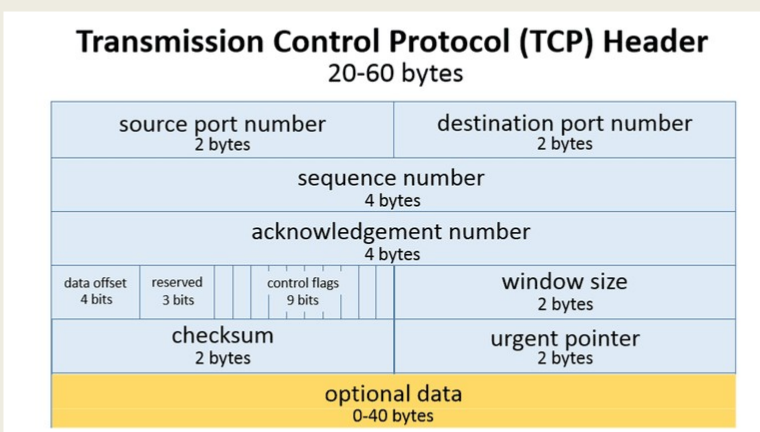
- New services can be built atop of existing layers to reuse low level implementation
- Abstractions make it easier reuse existing layers which already implement communication basics

HOW A NETWORK PACKET IS BUILT



October 10, 2017 | TCSS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma | L4.21

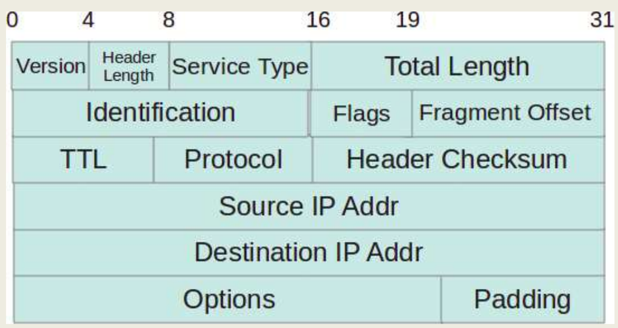
TCP HEADER



October 10, 2017 | TCSS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma | L4.22

IP HEADER

- Source / Destination IP Addr
- IPv4: 32bits / 4 bytes
- IPv6: 128bits / 16 bytes



TRANSMISSION CONTROL PROTOCOL (TCP)

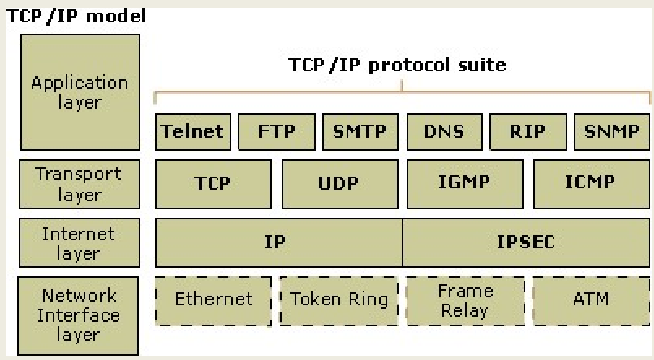
- TCP provides easy to use API
- API supports: setup, tear down of connection(s)
- API supports: sending and receiving of messages
- TCP preserves ordering of transferred data
- TCP detects and corrects lost data

- But TCP is “protocol” agnostic
  - E.g. language agnostic

- What are we going to say?

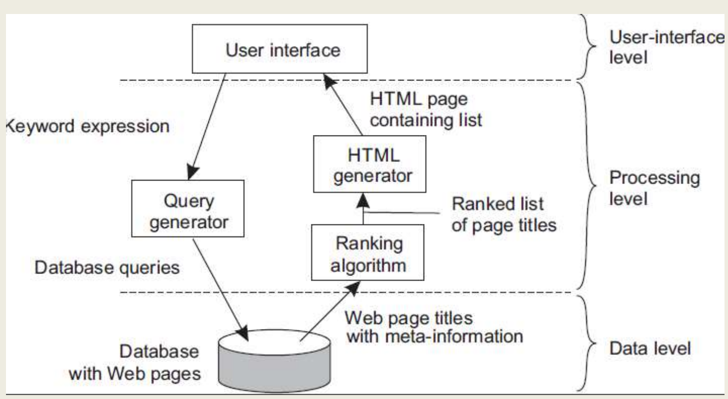
COMMON APPLICATION LAYER PROTOCOLS

- Telnet, FTP, TFTP, HTTP, DHCP, DNS, NTP, POP, RTP, SMTP, Telnet, RPC, LDAP



APPLICATION LAYERING

- Distributed application example: Internet search engine



APPLICATION LAYERING

- Three logical layers of distributed applications
  - The data level
  - Application interface level
  - The processing level

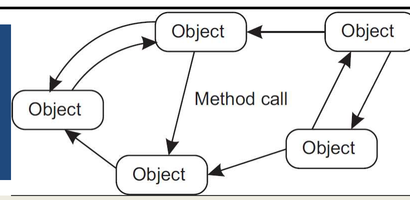
APPLICATION LAYERING

- Three logical layers of distributed applications
  - The data level (M)
  - Application interface level (V)
  - The processing level (C)
- Model view controller architecture – distributed systems
  - Model – database - handles data persistence
  - View – user interface - also includes APIs
  - Controller – middleware / business logic

OBJECT-BASED ARCHITECTURES

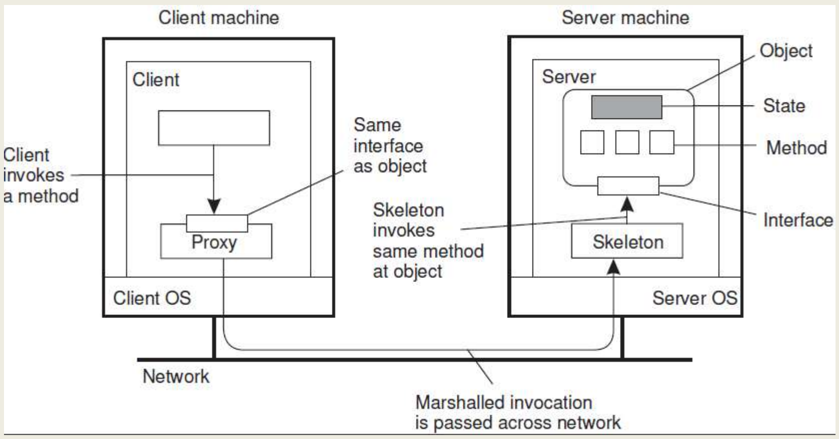
- Enables loose and flexible component organization
- Objects == components
- Enable distributed node interaction via function calls over the network
- Began with C - Remote Procedure Calls (RPC)
  - Straightforward: package up function inputs, send over network, transfer results back
  - Language independent
  - In contrast to web services, RPC calls originally were more intimate in nature
  - Procedures more "coupled", not as independent
  - The goal was not to decouple and widgetize everything

OBJECT-BASED ARCHITECTURES - 2



- Distributed objects Java- Remote Method Invocation (RMI)
  - Adds object orientation concepts to remote function calls
  - Clients bind to proxy objects
  - Proxy provide an object interface which transfers method invocation over the network to the remote host
- How do we replicate objects?
  - Object marshalling – serialize data, stream it over network
  - Unmarshalling- create an object from the stream
  - Unmarshall local object copies on the remote host
  - JSON, XML are some possible data formats

DISTRIBUTED OBJECTS



DISTRIBUTED OBJECTS - 2

- A counterintuitive features is that state is not distributed
- Each "remote object" maintains its own state
- Remote objects may not be replicated
- Objects may be "mobile" and move around from node to node
  - Common for data objects
- For distributed (remote) objects consider
  - Pass by value
  - Pass by reference

SERVICE ORIENTED ARCHITECTURE

- Services provide always-on encapsulated functions over the internet/web
- Leverage redundant cloud computing infrastructure
- Services may:
  - Aggregate multiple languages, libraries, operating systems
  - Include (wrap) legacy code
- Many software components may be involved in the implementation
  - Application server(s), relational database(s), key-value stores, in memory-cache, queue/messaging services

SERVICE ORIENTED ARCHITECTURE - 2

- Are more easily developed independent and shared vs. systems with distributed object architectures
- Less coupling
- An error while invoking a distributed object may crash the system
- An error calling a service (e.g. mismatching the interface) generally does not result in a system crash

RESOURCE BASED ARCHITECTURES

- Motivation:
  - Increasing number of services available online
  - Each with specific protocol(s), methods of interfacing
  - Connecting services w/ different protocols → integration nightmare
- Need for standardization of interfaces
  - Make services/components more pluggable
  - Easier to adopt and integrate
  - Common architecture

REST SERVICES

- Representational State Transfer (REST)
- Built on HTTP
- Four key characteristics:
  1. Resources identified through single naming scheme
  2. Services offer the same interface
     - Four operations: GET PUT POST DELETE
  3. Messages to/from a service are fully described
  4. After execution server forgets about client
     - Stateless execution

HYPERTEXT TRANSPORT PROTOCOL (HTTP)

- An ASCII-based request/reply protocol for transferring information on the web
- HTTP request includes:
  - request method (GET, POST, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - headers—extra info regarding transfer request
- HTTP response from server
  - Protocol version & status code →
  - Response headers
  - Response body

HTTP status codes:
2xx — *all is well*
3xx — *resource moved*
4xx — *access problem*
5xx — *server error*

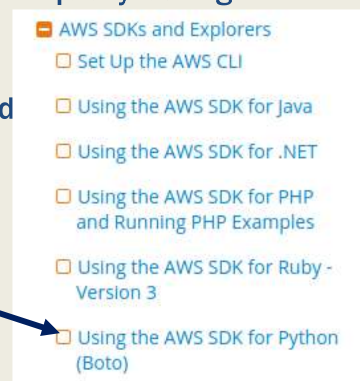
REST-FUL OPERATIONS

Operation	Description	
PUT	Create a new resource	(C)reate
GET	Retrieve state of a resource in some format	(R)ead
POST	Modify a resource by transferring a new state	(U)pdate
DELETE	Delete a resource	(D)elete

- Resources often implemented as objects in OO languages
- REST is weak for tracking state
- Generic REST interfaces enable ubiquitous *"so many"* clients

EXAMPLE: AMAZON S3

- Amazon S3 offers a REST-based interface
- Requires signing HTTP authorization header or passing authentication parameters in the URL query string
- REST: GET/PUT/POST/DELETE
- SOAP: 16 operations, moving toward deprecation
- Python boto ~50 operations (SDK for Python)
- SDKs for other languages



REST - 2

- Defacto web services protocol
- Requests made to a URI – uniform resource identifier
- Supersedes SOAP – Simple Object Access Protocol
- Access and manipulate web resources with a predefined set of stateless operations (known as web services)
- Responses most often in JSON, also HTML, ASCII text, XML, no real limits as long as text-based
- curl – generic command-line REST client: https://curl.haxx.se/

```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions  name ="DayOfWeek"
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="date" type="xsd:date"/>
  </message>
  <message name="DayOfWeekResponse">
    <part name="dayOfWeek" type="xsd:string"/>
  </message>
  <portType name="DayOfWeekPortType">
    <operation name="GetDayOfWeek">
      <input message="tns:DayOfWeekInput"/>
      <output message="tns:DayOfWeekResponse"/>
    </operation>
  </portType>
  <binding name="DayOfWeekBinding" type="tns:DayOfWeekPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetDayOfWeek">
      <soap:operation soapAction="getdayofweek"/>
      <input>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
        namespace="http://www.roguewave.com/soapworx/examples"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="DayOfWeekService" >
    <documentation>
      Returns the day-of-week name for a given date
    </documentation>
    <port name="DayOfWeekPort" binding="tns:DayOfWeekBinding">
      <soap:address location="http://localhost:8090/dayofweek/DayOfWeek"/>
    </port>
   </service>
</definitions>
```

L4.41

```
// REST/JSON
// Request climate data for Washington

{
 "parameter": [
  {
    "name": "latitude",
    "value":47.2529
  },
  {
    "name": "longitude",
    "value":-122.4443
  }
  ]
}
```

L4.42

QUESTIONS

October 10, 2017

TCSS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma

L4.43

EXTRA SLIDES

44