MD-SAL Application Development

MD-SAL Java Background, Environment, and Tools

Software Requirements

- JDK 1.7+
  - Set JAVA_HOME
- Maven 3.1+
  - Set MAVEN_HOME
  - repository: Where dependencies are downloaded and placed
  - settings.xml: Points Maven to where repositories are located
- Git
  - Install Git Client, e.g., Source Tree
- IDE
  - Intellij IDEA
  - Eclipse

Maven

```
<profile>
  <id>opendaylight-release</id>
  <repositories>
    <repository>
      <id>opendaylight-mirror</id>
      <name>opendaylight-mirror</name>
      <url>http://nexus.opendaylight.org/content/repositories/public/</url>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>opendaylight-mirror</id>
      <name>opendaylight-mirror</name>
      <url>http://nexus.opendaylight.org/content/repositories/public/</url>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
.................
 <activeProfiles>
    <activeProfile>opendaylight-release</activeProfile>
    <activeProfile>opendaylight-snapshots</activeProfile>
 </activeProfiles>
```

OPEN DAYLIGHT

Maven

Some common pre-defined goals:

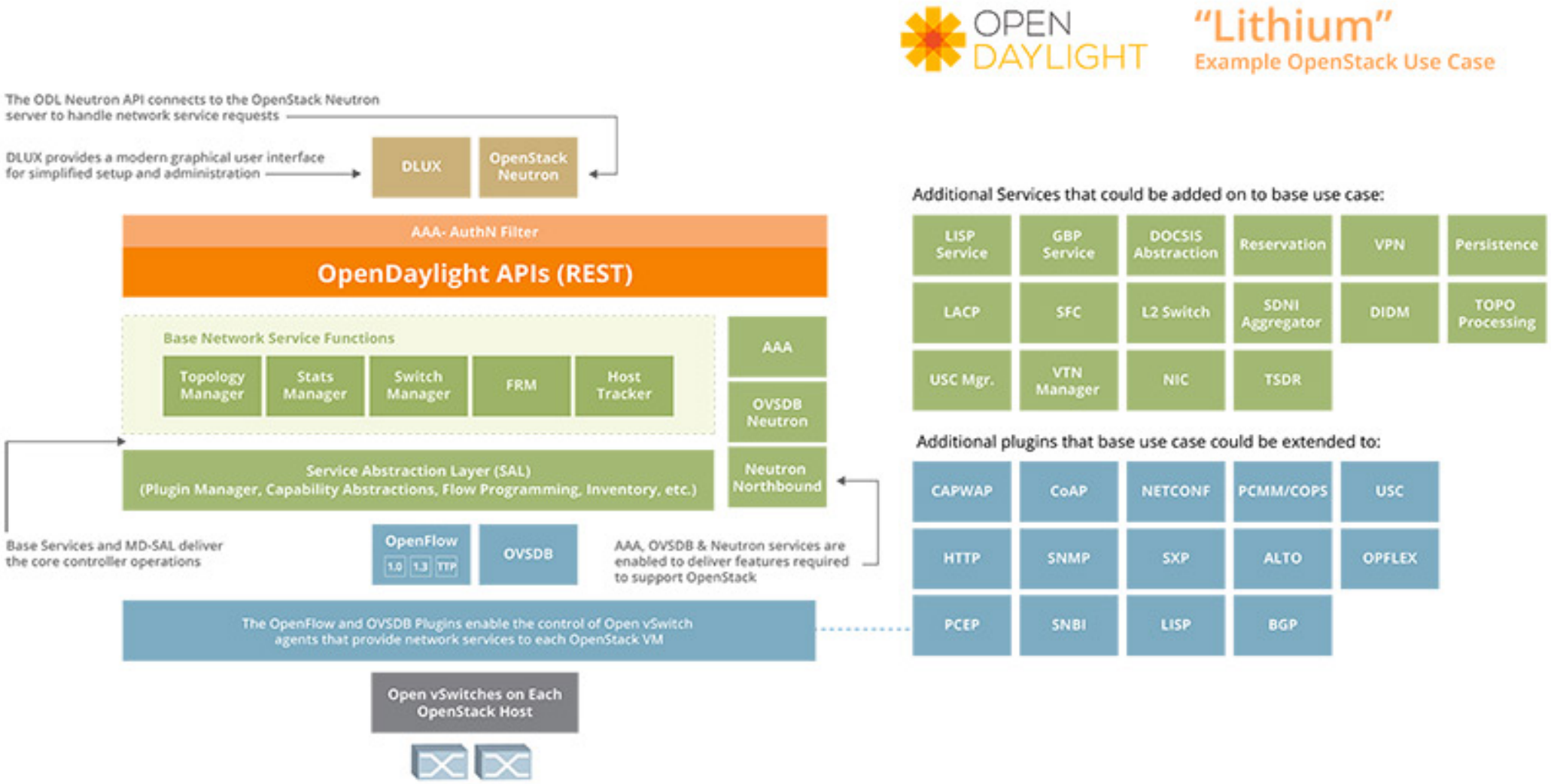
- clean: clean up working directory of files that were created at build time.
- install: install the built artifact(s) as directed
- -DskipTests: don’t running the tests for this build

maven.apache.org

set MAVEN_OPTS=-Xmx512m -XX:MaxPermSize=128m

ODL Architecture

ODL Architecture



OPEN DAYLIGHT

Controller Platform/Framework

- 采用了OSGI Framework，实现了模块化和可扩展化
- SAL是Controller Platform的最核心模块
- 由运行在OSGI Framework上的Bundle实现
- 控制模块间数据交互，数据存取，API调用

Northbound REST

- 可扩展
- REST是以资源的角度观察整个网络，分布在各处的资源由URI确定
- 目的是为不同地址空间的应用提供接口; 应用可能是多样性的

Southbound Netty

- 南向接口支持多种协议; 这些协议模块以插件的方式动态挂在SAL上
- 南向接口使用Netty来管理底层的并发IO
- Netty是提供异步的、事件驱动的网络应用框架; 该框架健壮性、扩展性良好，而且还具有延时低、节省资源等特点

Plugin

- 通过SAL实现的功能模块
- 运行在ODL OSGI Framework上，与ODL控制器共享同一个JVM资源
- Southbound Plugin
  - 向SAL提供管理控制南向设备或服务的操作接口
- Northbound Plugin
  - 通过利用SAL所提供的北向服务API向应用提供统一的抽象服务和相应的API

Karaf

OSGI(Open Services Gateway Initiative)

- 通俗点说JAVA动态模块系统，定义了一套模块应用开发的框架
- OSGI容器允许把应用分成多个功能模块，通过依赖管理这些功能会更方便
- Bundle 是 OSGi 中的基本组件，其表现形式仍然为Java概念中传统的Jar

Karaf

- Karaf中引用了Feature的概念; Feature是符合某个功能特性的bundle集的部署描述

  A feature describes an application as:
  - a name
  - a version
  - an optional description (eventually with a long description)
  - a set of bundles
  - optionally a set configurations or configuration files
  - optionally a set of dependency features

Karaf

The directory layout of a Karaf installation is as follows:

- /bin: control scripts to start, stop, login, ...
- /demos: contains some simple Karaf samples
- /etc: configuration files
- /data: working directory
- /cache: OSGi framework bundle cache
- /generated-bundles: temporary folder used by the deployers
- /log: log files
- /deploy: hot deploy directory
- /instances: directory containing instances
- /lib: contains libraries
- /lib/boot: contains the system libraries used at Karaf bootstrap
- /lib/endorsed: directory for endorsed libraries
- /lib/ext: directory for JRE extensions
- /system: OSGi bundles repository, laid out as a Maven 2 repository

Karaf Commands

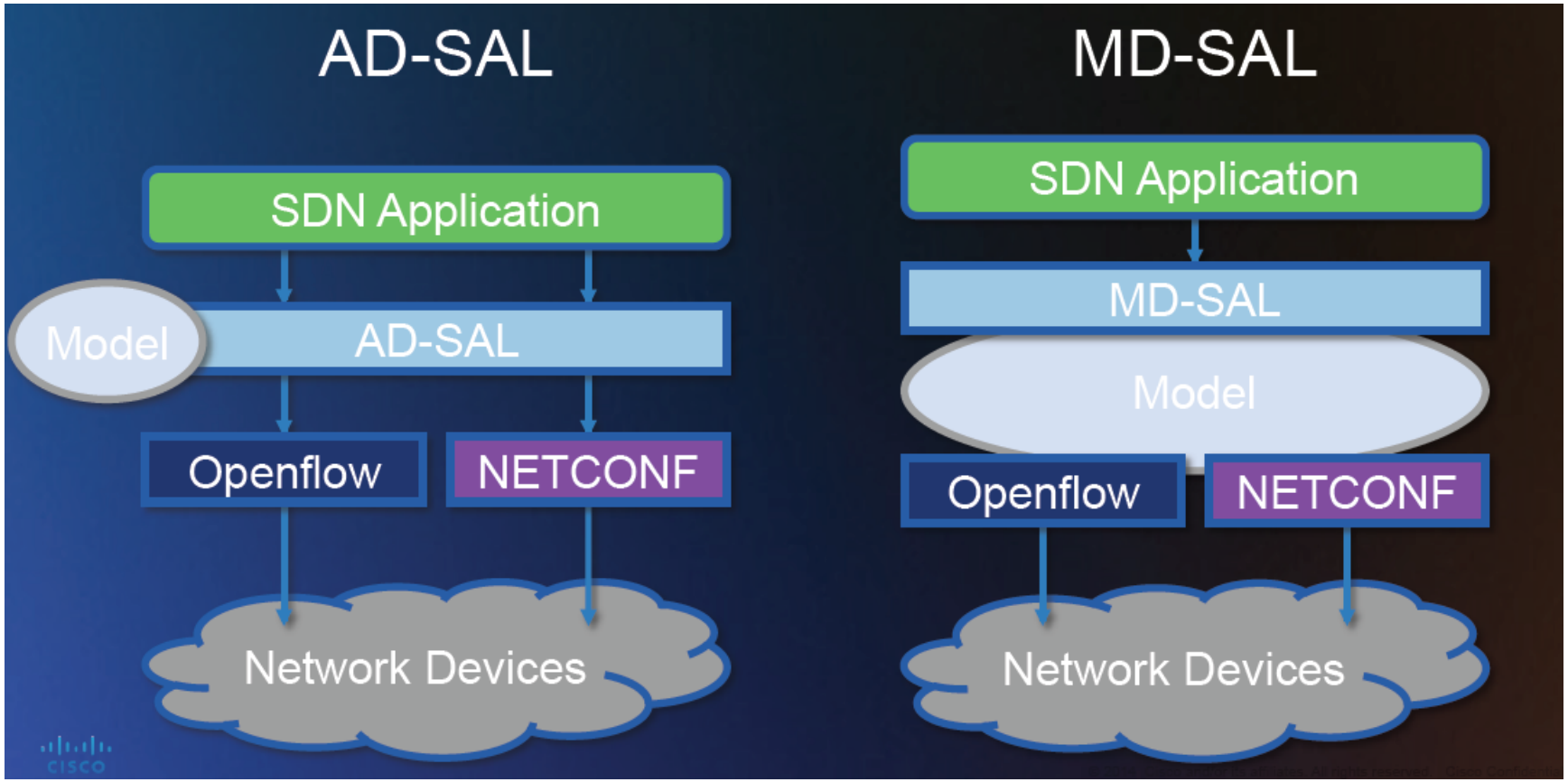
- Logging
  - log:display - Displays the entire log
  - log:tail - Continuously displays the last lines of the log
- Features
  - feature:list - Lists all the features known to the controller (installed or not)
  - feature:install - Installs a feature
- Repository
  - repo:mvn... - Specifies a repository in which Karaf will look for features

karaf.apache.org/manual/latest/commands/commands.html

MD-SAL

Why MD-SAL: Model-based

- Operations based on model:
  - Applications interact with the model; model interacts with the network.
  - This abstraction allows for any (or multiple) southbound plugin(s).



OPEN DAYLIGHT

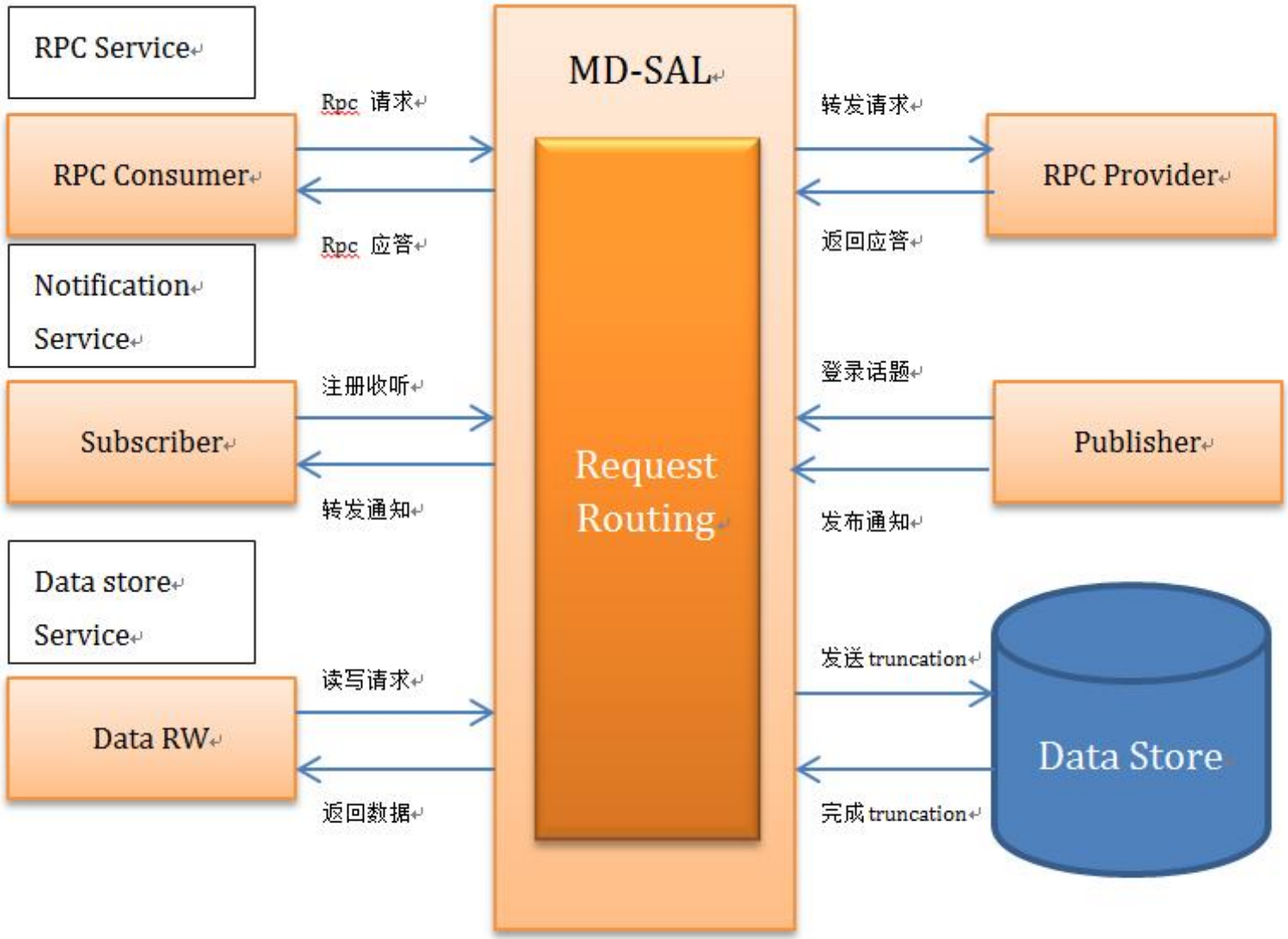
Overview

- 简单的说，MD-SAL是一套基础设施服务，为Apps和Plugins开发提供通用支持
- 提供Request Routing和用来实现抽象服务和相应API的基础框架
  - 抽象服务和相应API是由各个Plugin通过Yang Model和Service来定义
  - Yang Tools Plugin通过各个Plugin的Model定义来自动生成API、Service Interface和相应Java代码
  - 开发者通过实现自动生成的Service Interface来实现具体的API和服务内容
  - Plugin通过MD-SAL和生成的API（RPC, Notification）、DataStore去利用其他各个Plugin的服务和数据
- 所有功能模块的信息交互，数据存储调用都通过MD-SAL完成

Function

MD-SAL的主要功能就是管理基于Yang Model定义的各种Plugin。

- RPC：提供服务的远程Call接口
- Notification：提供Notification收听，发行等功能
- Data Store：提供数据存储，读取，Transaction等功能
- Request Routing：提供请求路由功能，把外部的请求传送到正确的Plugin和Node Instance处理
  Node instance是Yang结构树上的节点实例。
- RestConf Subsystem：自动定义和创建RestConf API的Plugin
- Config subsystem：提供统一的配置文件管理功能的Plugin



OPEN DAYLIGHT

MD-SAL Plugin

根据Plugin和MD-SAL的关联方式，Plugin分为如下两种

- BA（Binding-Aware）

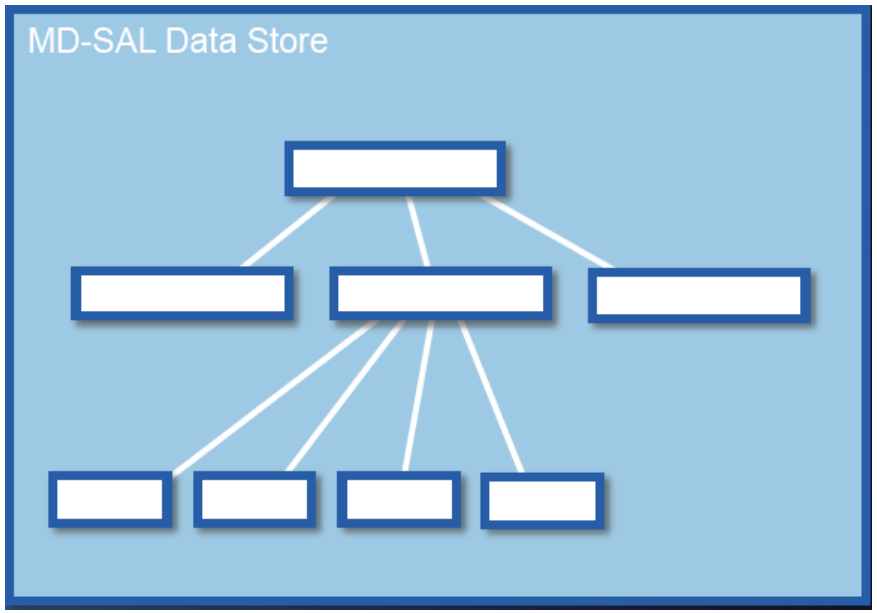
指使用根据Yang Model定义而自动生成的Java Bindings的Plugin

- BI（Binding-independent）

与BA相反，不依赖于Java Bindings的Plugin

YANG Models

YANG Data Tree

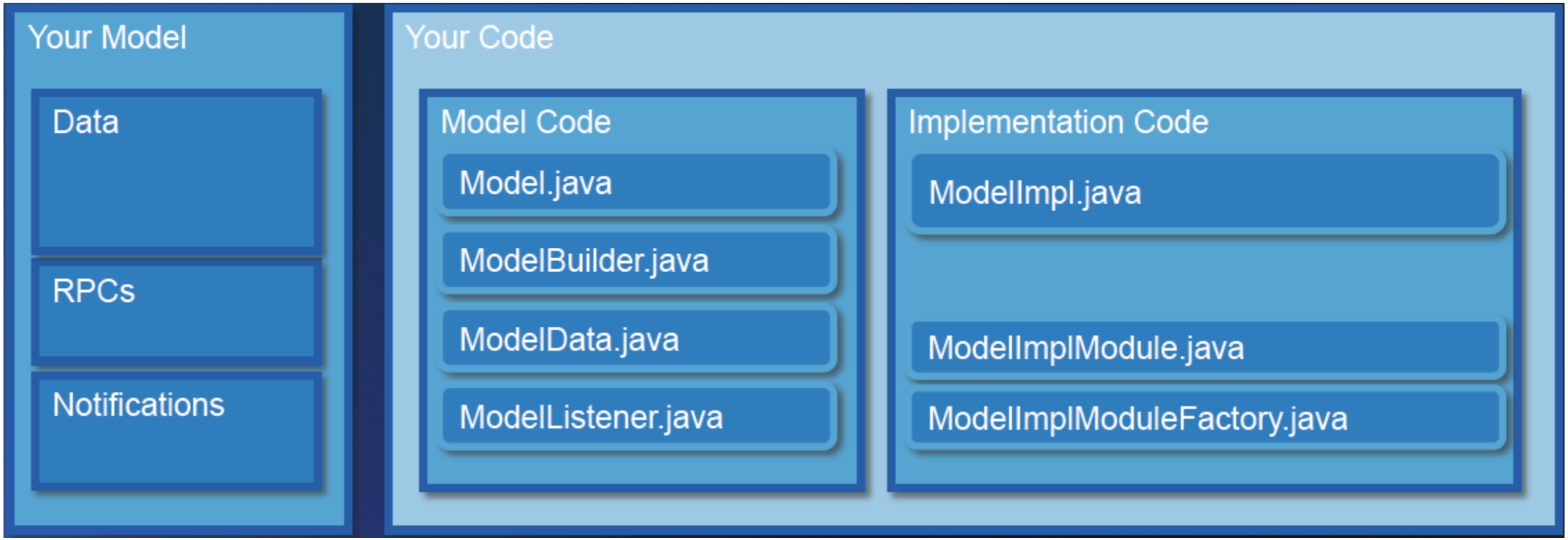


Why is this important?

- **Storage:** You will be *storing* data into the MD-SAL Data Store using the Data Broker
- **Event notifications:** You will be *listening* for notifications indicating that somebody or something has changed data associated with your model.

OPEN DAYLIGHT

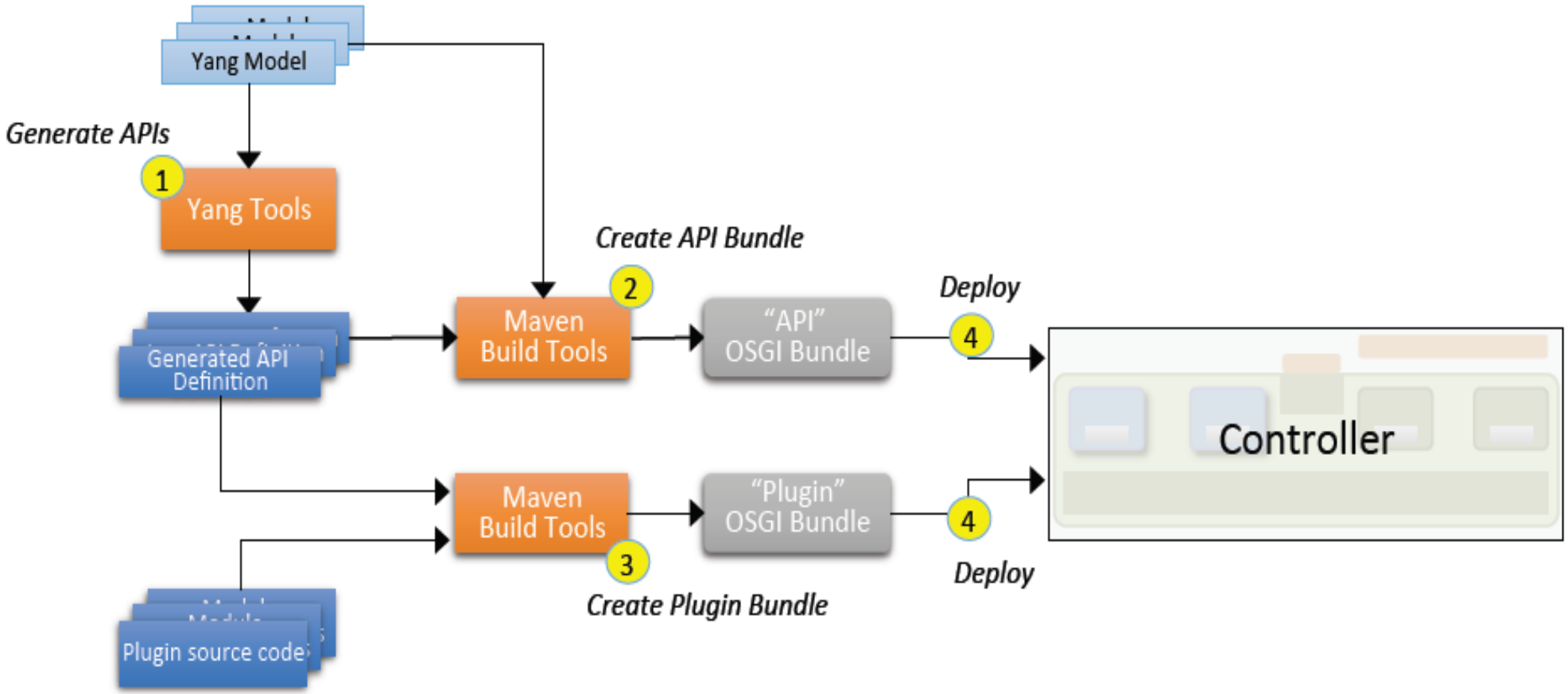
Model-generated Code



YANG Tools

How MD-SAL Works: Auto-generated Model and API

- Define you application’s model using YANG
- Compile model
- Model code auto-generated
- REST API code auto-generated
- You implement business logic



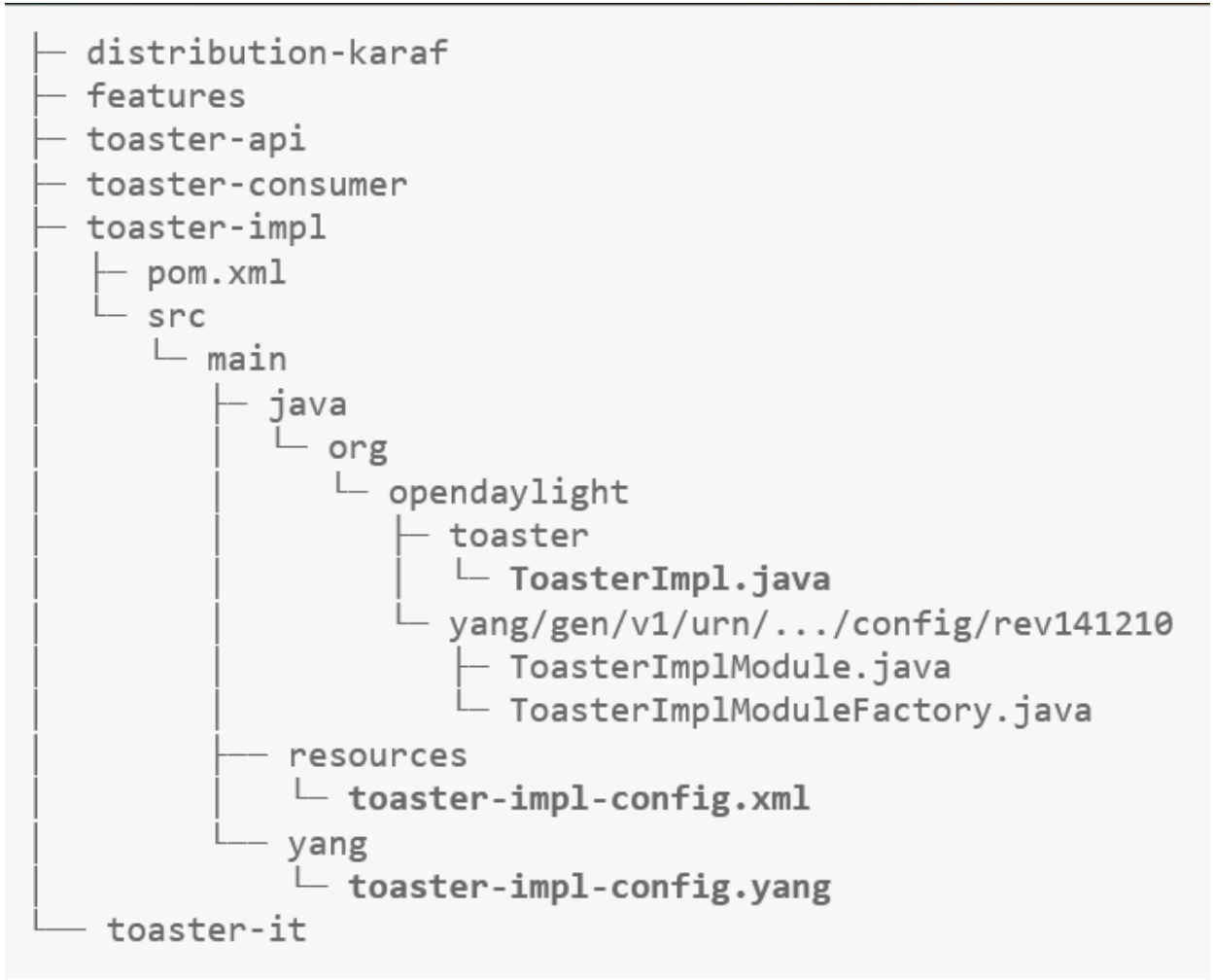
MD-SAL Core Tutorials

First Sample Application

Connecting Your Application to Controller

Three connection points:

- **Model:** YANG model for 'impl' informs controller of your configuration files.
- **Configuration:** Configuration file for 'impl' identifies the MDSAL broker with which you will be working.
- **Code:** Your 'impl' code will implement methods to initialize our Toaster application
  - **createInstance()** in ToasterImplModule class
  - **onSessionInitiated()** in ToasterImpl class

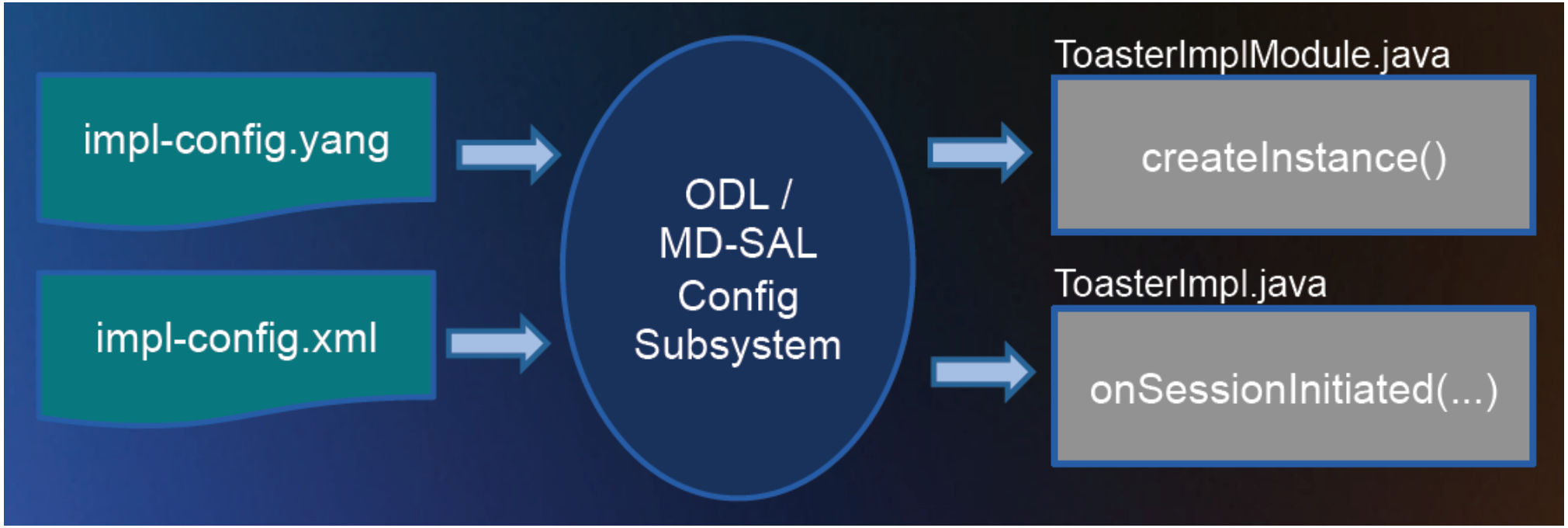


OPEN DAYLIGHT

Wiring Application to MD-SAL

- **YANG:** Defines wiring details
- **Config:** Defines dependencies
- **Code:** Handles initialization process

- **createInstance:** called when our module is created.
- **onSessionInitiated:** called when our provider is created and registered



OPEN DAYLIGHT

Initializing Your Application

ToasterImplModule:

- **createInstance():** Override ‘createInstance’ in ToasterImplModule
- **Called by controller:** ‘createInstance’ called by controller when app starts
- **Create your ‘provider’ :** Create your provider, i.e. your toaster implementation, called ‘ToasterImpl’
- **Register your provider:** Register your provider with controller

ToasterImplModule.java

```
@Override
public java.lang.AutoCloseable createInstance() {

    ToasterImpl provider = new ToasterImpl();
    getBindingAwareBrokerDependency().registerProvider( provider, null );

    return provider;
}
```

OPEN DAYLIGHT

Initializing Your Application

ToasterImpl:

- **onSessionInitiated:** Override ‘onSessionInitiated’
- **Called by controller:** Called by controller when your provider starts up and is registered in ‘createInstance()’
- For sample application: just log a ‘hello world’ message

ToasterImpl.java

```
@Override
public void onSessionInitiated( ProviderContext context ) {

    LOG.info("Hello World!");

}
```

Yang

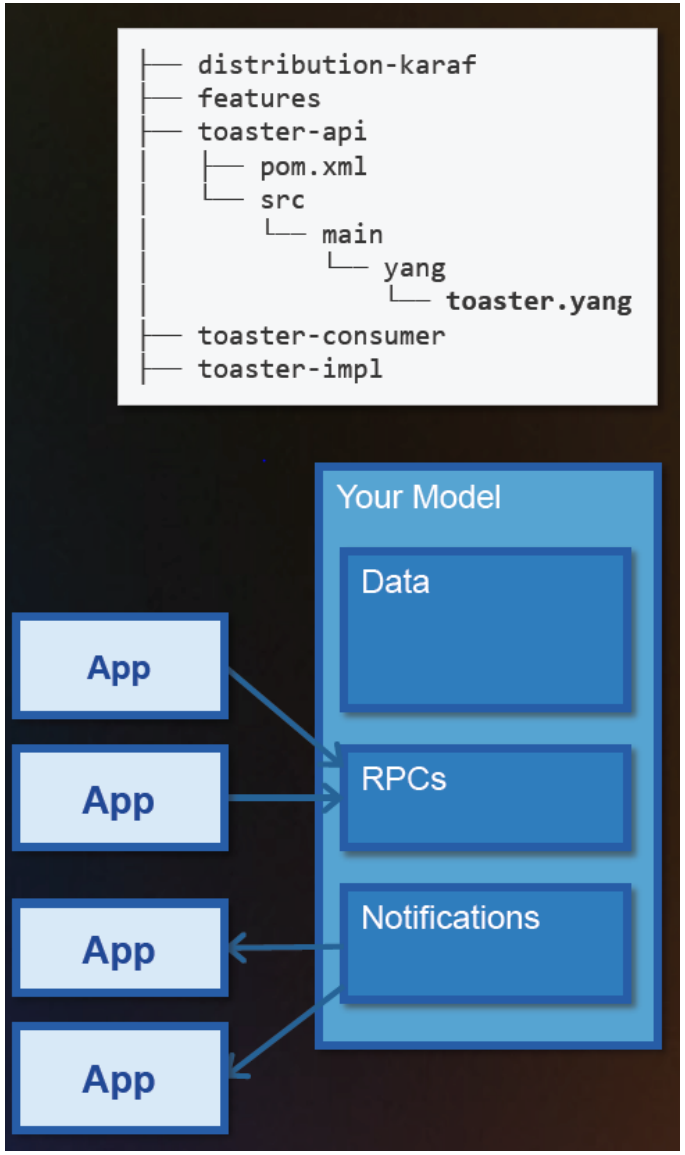
YANG Definitions

YANG - Defining your application:

- **Directory - toaster-api**: where the YANG definition of your application' s model resides
- **src/main/yang/toaster.yang**: the actual YANG file that holds the application details.

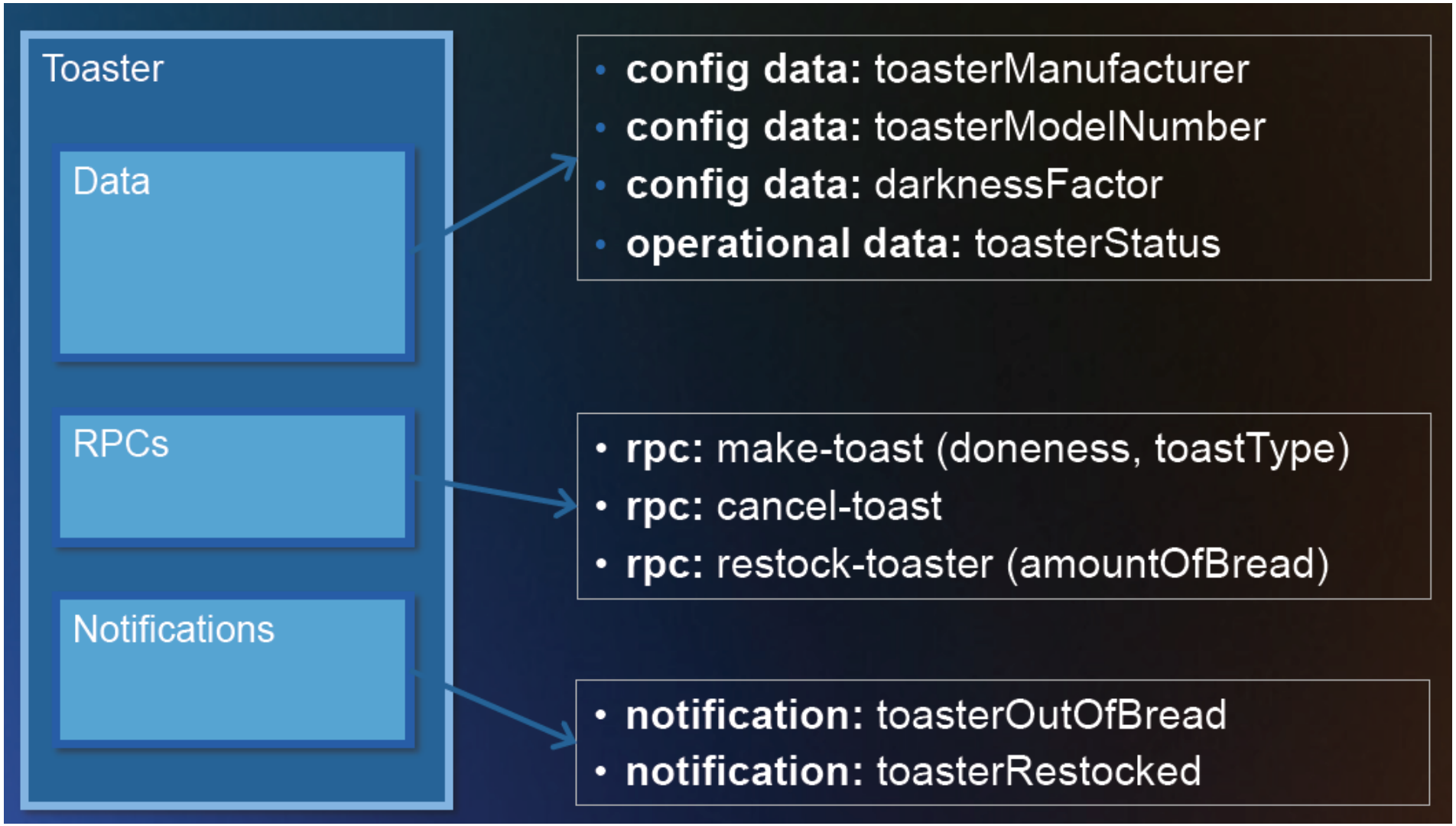
YANG - Components:

- **Data:** defines the information that comprises your application. Identification, status, lists of stored items, etc.
- **RPCs:** Operations (APIs) that can be invoked on your model, such as tasks to be performed, status information to be retrieved, etc.
- **Notifications:** Messages that will be sent by this model object to registered listeners, informing them of state changes, new items created, deleted, changed, etc.



OPEN DAYLIGHT

YANG Toaster Definition



OPEN DAYLIGHT

Features Definition for our Model

Feature definition in features.xml

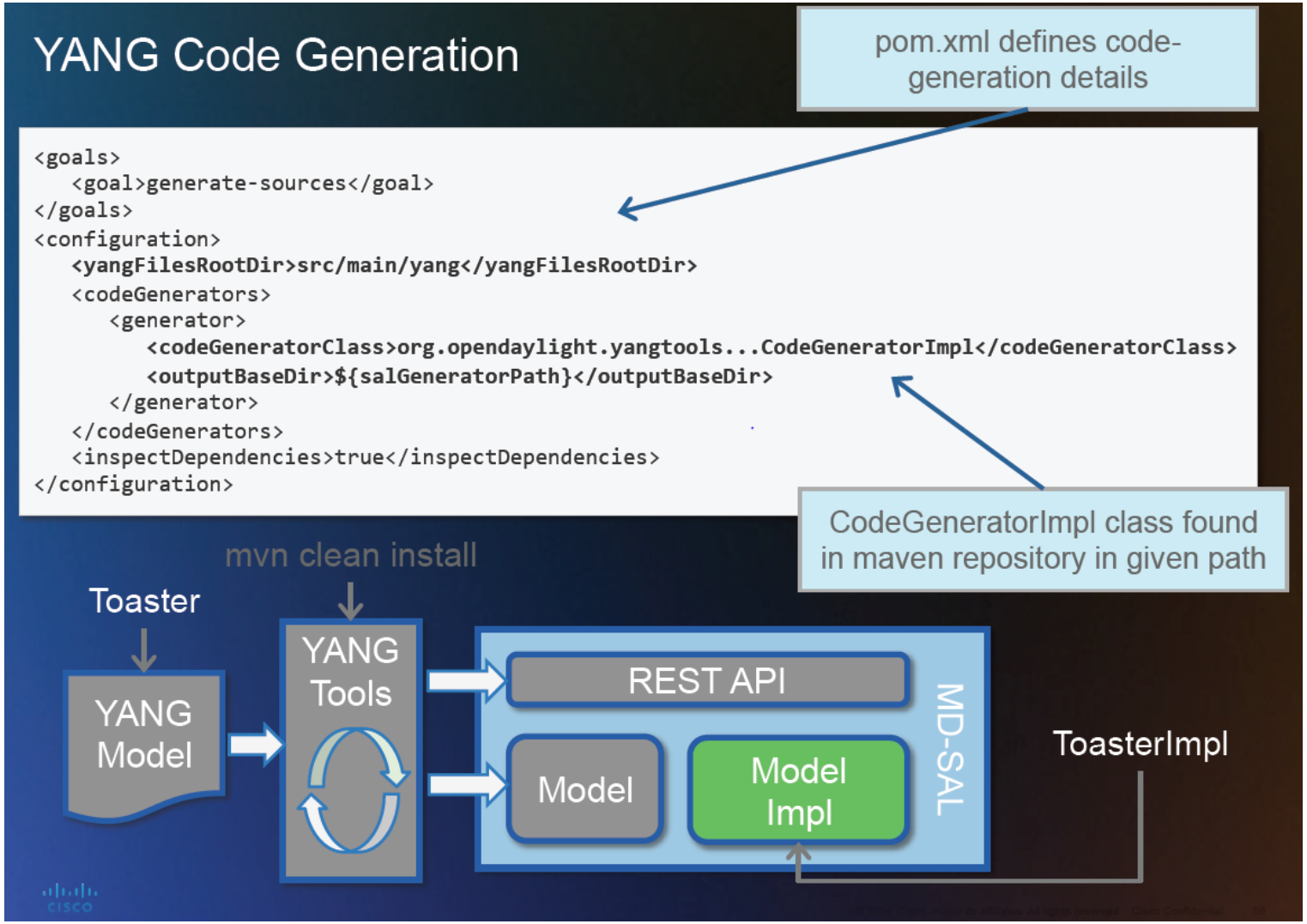
- **Name:** Defines feature ‘odl-toaster-api’
- **Dependencies:** Includes dependencies on:
  odl-yangtools-common
  odl-yangtools-binding
  odl-restconf
- **Bundle:** A bundle is built from files in this folder

```
<feature name='odl-toaster-api' version='${project.version}' description=...'>

     <feature version='${yangtools.version}'>odl-yangtools-common</feature>
     <feature version='${yangtools.version}'>odl-yangtools-binding</feature>
     <feature version='${controller.restconf.version}'>odl-restconf</feature>

     <bundle>mvn:org.opendaylight.toaster/toaster-api/${project.version}</bundle>

</feature>
```

OPEN DAYLIGHT

Reading Model Data

YANG Code Generation



Generated Code

toaster-api/target/generated-sources/sal: Generated code placed in this directory

- ***.java:** Java files created out of YANG model definitions
- **Container:** YANG definition of *toaster* becomes Java interface *Toaster*
  - Includes interface for getting *ToasterStatus, Manufacturer, ModelNumber*
  - Includes definition for possible ToasterStatus values (*Up, Down*)
- **Identities:** YANG definition for various *toast-types* become abstract classes
  - *wheat-bread* becomes abstract class *WheatBread*
  - *whole-bread* becomes abstract class *WholeBread*
  - *frozen-bagel* becomes abstract class *FrozenBagel*
  - etc.
- **RPCs** and **Notifications:** Covered later

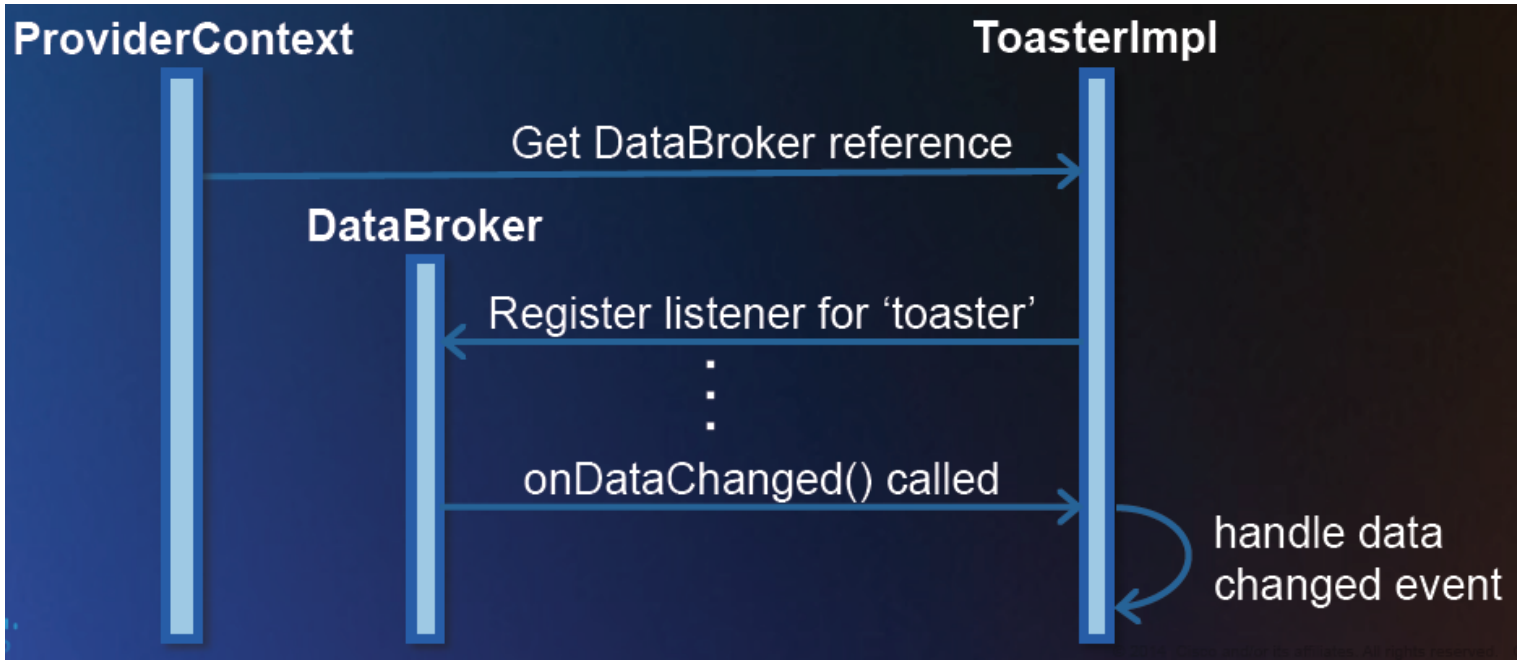
Getting Data Change Notifications from MD-SAL

Receiving notifications of changes to data in MD-SAL data store

- **Data Broker:** The MD-SAL ‘DataBroker’ is your interface to the MD-SAL data store.

Steps for receiving data change notifications

- **DataBroker reference:** get reference to DataBroker
- **Register listener:** register listener for changes to your model objects
- **Handle notifications:** receive notification via the *onDataChanged()* method



OPEN DAYLIGHT

Registering Listener

Instance Identifier:

• Create Instance Identifier (‘IID’) for your model object (Toaster.class)

```
public static final InstanceIdentifier<Toaster> TOASTER_IID =
                                                InstanceIdentifier.builder(Toaster.class).build();
```

Register Listener with DataBroker: *registerDataChangeListener(...)*

- Data type: *CONFIGURATION* or *OPERATIONAL*
- Instance Identifier: IID of model item in YANG data tree (Toaster)
- Location: in what object is this listener (‘this’, i.e. your ToasterImpl)
- Scope: *BASE* (this object only), *ONE* (this object plus one level below), or *SUBTREE* (this object plus all of its descendants)

```
dcReg = dataService.registerDataChangeListener( LogicalDatastoreType.CONFIGURATION,
                                                TOASTER_IID,
                                                this,
                                                DataChangeScope.SUBTREE );
```

Data Change Notifications

Handling data change notifications

1. **DataChangeEvent:** You receive an AsyncDataChangeEvent ('change')

2. **Get new data:** Retreive changes to your model objects ('getUpdatedSubtree')

3. **Coerce data into Toaster:** Assign updated subtree to your model object ('toaster')

```
@Override
public void onDataChanged( final AsyncDataChangeEvent<InstanceIdentifier<?>, DataObject> change ) {

    DataObject dataObject = change.getUpdatedSubtree();

    if( dataObject instanceof Toaster ) {
        Toaster toaster = (Toaster) dataObject;
        LOG.info("onDataChanged - new Toaster config: {}", toaster);
    }
    else {
        LOG.warn("onDataChanged - not instance of Toaster {}", dataObject);
    }

}
```

1

2

3

4

Writing Model Data

Transactions

General process for writing data to the model using transactions

1. **Data:** Create your data using YANG-generated builder and setter methods
2. **Create:** Create the empty write transaction using data broker service
3. **Put:** Populate the write transaction with your data
4. **Submit:** Submit your transaction

```
Toaster toaster = new ToasterBuilder().setToasterManufacturer( TOASTER_MANUFACTURER )
                                      .setToasterModelNumber( TOASTER_MODEL_NUMBER )
                                      .setToasterStatus( ToasterStatus.Up )
                                      .build();

WriteTransaction tx = dataService.newWriteOnlyTransaction();

tx.put(LogicalDatastoreType.OPERATIONAL, TOASTER_IID, toaster);

tx.submit();
```

1 2 3 4

OPEN DAYLIGHT

Asynchronous Operation

General process for asynchronous operation:

1. Create and populate transaction, but don't *submit*

2. Submit transaction using Futures class

3. Create anonymous class for asynchronous completion of task

    - Override *onSuccess* and *onFailure* in *Futures.addCallback(...)*

```
Toaster toaster = new ToasterBuilder(). ... .build();
WriteTransaction tx = dataService.newWriteOnlyTransaction();
tx.put(LogicalDatastoreType.OPERATIONAL, TOASTER_IID, toaster);

Futures.addCallback( tx.submit(), new FutureCallback<Void>() {
            @Override
            public void onSuccess(final Void result) {
                LOG.info("initToasterOperational: transaction succeeded");
            }
            @Override
            public void onFailure(final Throwable t) {
                LOG.error("initToasterOperational: transaction failed");
            }
} );
```

1

2

3

Synchronous vs Asynchronous

Write transaction (synchronous):

- Call ‘submit’ method

```
tx.submit();
```

Write transaction (asynchronous):

- Call ‘addCallback’ method of ‘Futures’ static class
- Pass in ‘tx.submit()’, and create a new ‘FutureCallback’ object
- Override ‘onSuccess’ and ‘onFailure’ in FutureCallback object

```
Futures.addCallback( tx.submit(), new FutureCallback<Void>() {
            @Override
            public void onSuccess(final Void result) {
                LOG.info("initToasterOperational: transaction succeeded");
            }
            @Override
            public void onFailure(final Throwable t) {
                LOG.error("initToasterOperational: transaction failed");
            }
} );
```

OPEN DAYLIGHT

Anonymous Class: Alternative

Anonymous classes:

- Purpose is to simplify, but often degrades readability
- Alternative: make anonymous class a normal private inner class

```
Futures.addCallback( tx.submit(), new ToasterInitCallback() );

...

private class ToasterInitCallback implements FutureCallback<void> {

    @Override
    public void onSuccess(final Void result) {
       LOG.info("initToasterOperational: transaction succeeded");
    }
    @Override
    public void onFailure(final Throwable t) {
        LOG.error("initToasterOperational: transaction failed");
    }

}
```

RPCs

Defining RPCs in YANG

Defining RPCs in YANG

Toast RPC definitions

- In **toaster-api** project
- **make-toast, cancel-toast,** and **restock-toaster:** Defined as ‘rpc’ in YANG definition in our YANG mod el file in *toaster-api*
- **input { ... }:** Define inputs to the RPC calls in YANG

Toast RPC implementations

- In **toaster-impl** project
- **makeToast(...), cancelToast(),** and **restockToaster(...):** Methods in your ‘ToastImpl’ class
- **inputs:** auto-generated classes, e.g. ‘MakeToastInput’

```
rpc make-toast {
    input {
        leaf toasterDoneness {
            type uint32 {
                range "1 .. 10";
            }
            default '5';
        }
        leaf toasterToastType {
            type identityref {
                base toast:toast-type;
            }
            default 'wheat-bread';
        }
    }
}  // rpc make-toast

rpc cancel-toast {
}  // rpc cancel-toast

rpc restock-toaster {
    input {
        leaf amountOfBreadToStock {
            type uint32;
        }
    }
}  // rpc restock-toaster
```

OPEN DAYLIGHT

Defining RPCs in YANG

Tasks for implementing RPCs

- In **ToasterImpl** class, must *implement* auto-generated **ToasterService**

```
public class ToasterImpl implements ..., ToasterService, ... {
```

- **Register:** Must register our Toaster Service class with RPC registry

```
@Override
public void onSessionInitiated( ProviderContext session ) {

    // Register the RPC Service
    rpcReg = session.addRpcImplementation( ToasterService.class, this );

}
```

- **Methods:** Must implement our Toaster Service methods, e.g. **makeToast**

```
@Override
public Future<RpcResult<Void>> makeToast( final MakeToastInput input ) {
    LOG.info( "makeToast: {}", input );
    return Futures.immediateFuture( RpcResultBuilder.<Void> success().build() );
}
```

OPEN DAYLIGHT

About These RPC Implementations...

Framework complete

- We have created model objects in our YANG definition
- We have implemented methods for each RPC in our model definition
- **However:** Our RPC methods don't actually do anything yet, other than LOG a message and immediately return success to the caller. See below:

```
@Override
public Future<RpcResult<Void>> makeToast( final MakeToastInput input ) {

    LOG.info( "makeToast: {}", input );
    return Futures.immediateFuture( RpcResultBuilder.<Void> success().build() );

}
```

Putting it all Together

Atomic Variables

• **Concurrency:** Required for concurrency in modern systems with multiple processors, multiple cores, multiple independent caches

• **Lock-free:** Protection for single variable atomic operations without requiring explicit locking and synchronization functions

• **Replace '++', '--':** Allows for atomic increments using 'incrementAndGet', 'addAndGet', 'getAndIncrement', 'compareAndSet', etc. operations

```
private final AtomicLong amountOfBreadInStock = new AtomicLong( 100 );
private final AtomicLong toastsMade           = new AtomicLong( 0 );
...

private class MakeToastTask implements Callable<Void> {
    ...
    // Increment number of toasts made, decrement bread left in stock
    toastsMade.incrementAndGet();
    amountOfBreadInStock.getAndDecrement();
    ....
}
```

OPEN DAYLIGHT

Completing RPC Implementations (simplified version)

Process to get current toaster status, and create task to make toast

- **Read Toaster data:**

```
final ReadWriteTransaction tx = dataService.newReadWriteTransaction();

ListenableFuture<Optional<Toaster>> readFuture =
                      tx.read( LogicalDatastoreType.OPERATIONAL, TOASTER_IID );

final ListenableFuture<Void> commitFuture =
                  Futures.transform( readFuture, new ReadToasterDataFunction() );
```

- **Create task to make toast:**

```
Futures.addCallback( commitFuture, new MakeToastCallbackFunction() );
```

- **Return *futureResult*:**

```
futureResult.set( RpcResultBuilder.<Void>success().build() );
```

Notifications

- **Save notification service on session initiation**

```
private NotificationProviderService notificationService;

public void onSessionInitiated( ProviderContext session ) {
   ...
   this.notificationService =
                     session.getSALService( NotificationProviderService.class );
   ...
}
```

- **Send notification using service**

```
if( outOfBread() ) {

   notificationService.publish( new ToasterOutOfBreadBuilder().build() );

}
```

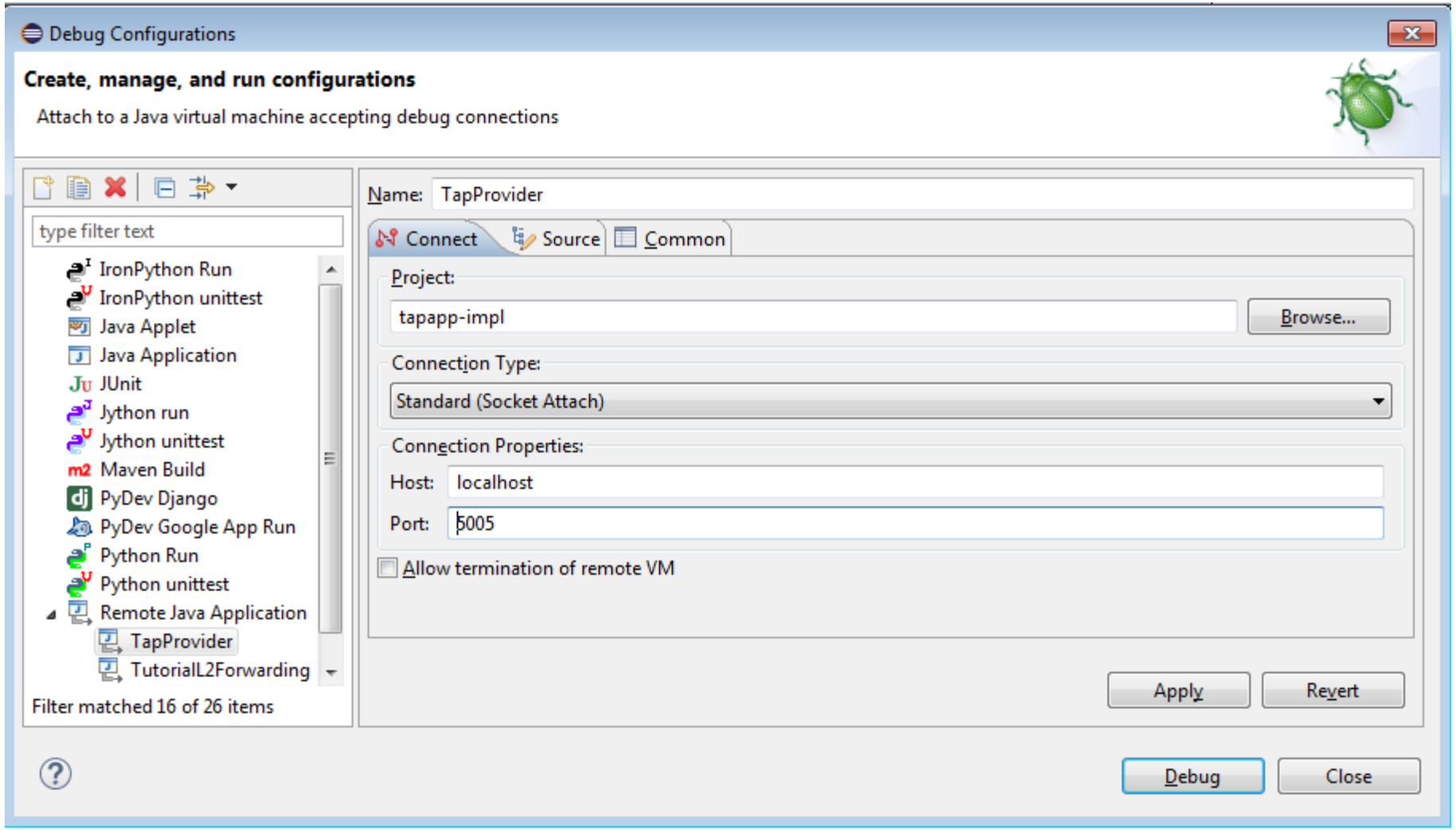
Debug & Release

Debug

%user-root%>%karaf_home%/bin/karaf -debug

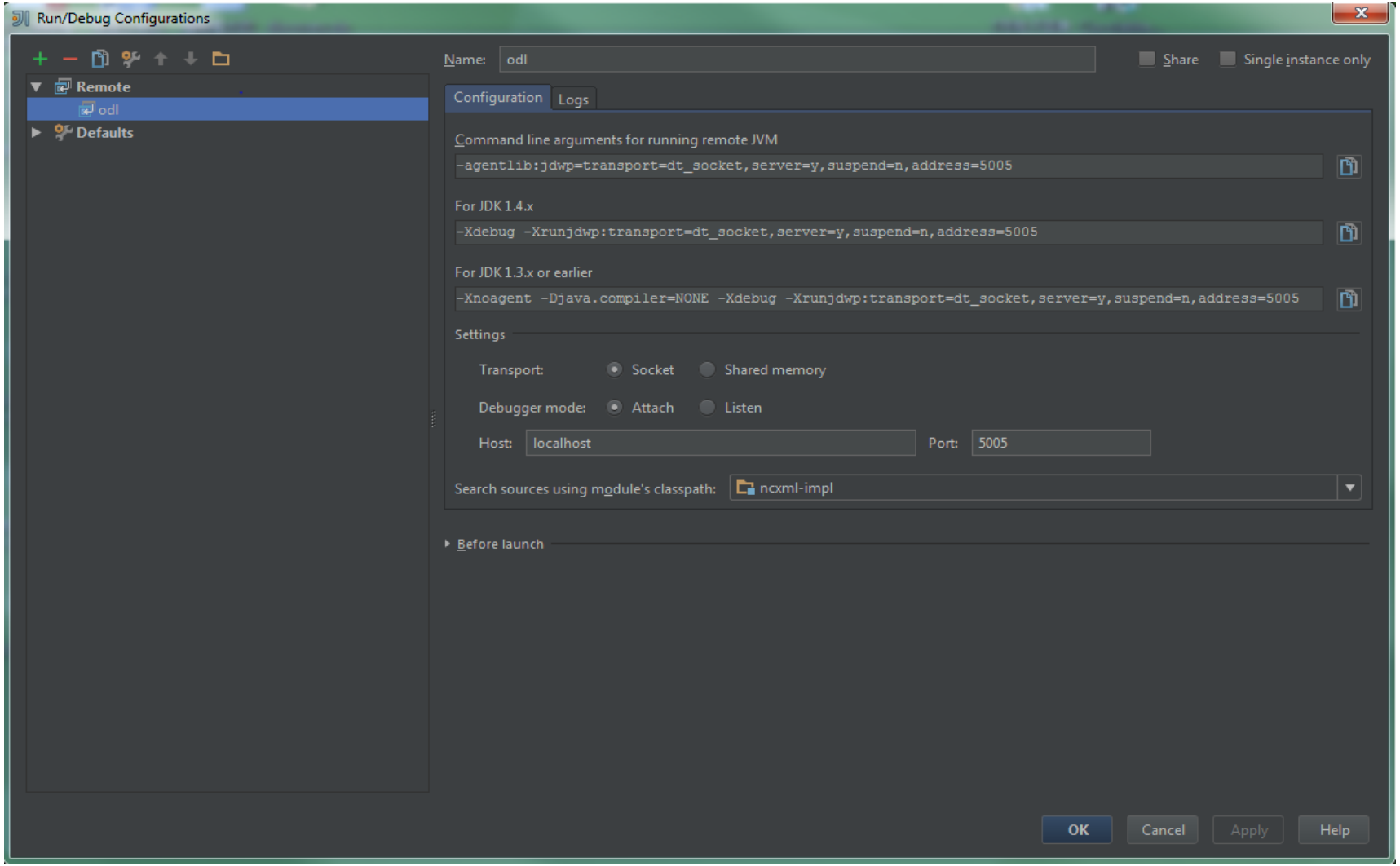
%user-root%>%karaf_home%/bin/karaf .bat -debug

Debug



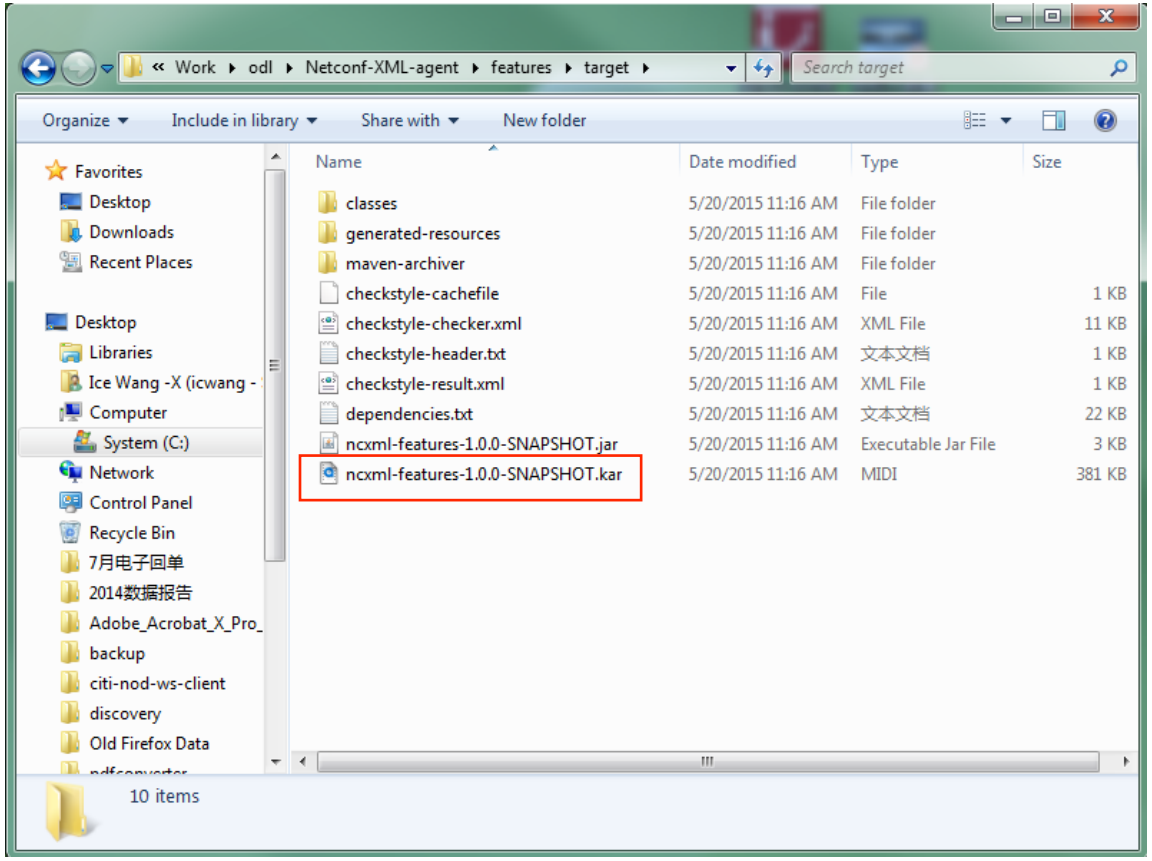
OPEN DAYLIGHT

Debug



OPEN DAYLIGHT

Release



1. Put xxxx.kar into karaf\target \assembly\deploy

2. In karaf console, use command line

   feature: install file:%file_path%

OPEN DAYLIGHT

Thank You