1. Medical Billing Dashboard Home 2
    1.1 Frontend Overview 2
    1.2 AWS Overview 21
        1.2.1 AWS SSO 22
        1.2.2 Updating products in the Xillum Service Catalog portfolio 23
        1.2.3 Cloud Trail Logs 24
        1.2.4 S3 Inventory 25
        1.2.5 New Customer Account Setup 26
            1.2.5.1 Enroll a new customer account 26
            1.2.5.2 Provision an enrolled customer account 27
            1.2.5.3 Service Catalog 28
            1.2.5.4 VPC 29
            1.2.5.5 MySQL RDS Database 31
            1.2.5.6 Cognito 33
            1.2.5.7 DNS & SSL Certificate 34
            1.2.5.8 Xillum REST API 36
            1.2.5.9 Xillum Frontend 36
            1.2.5.10 Secure S3 Upload Bucket 39
        1.2.6 To Be Deleted 40
    1.3 Database Overview 40
    1.4 PHP Script Recommendations 41

Medical Billing Dashboard Home

Welcome to your new space

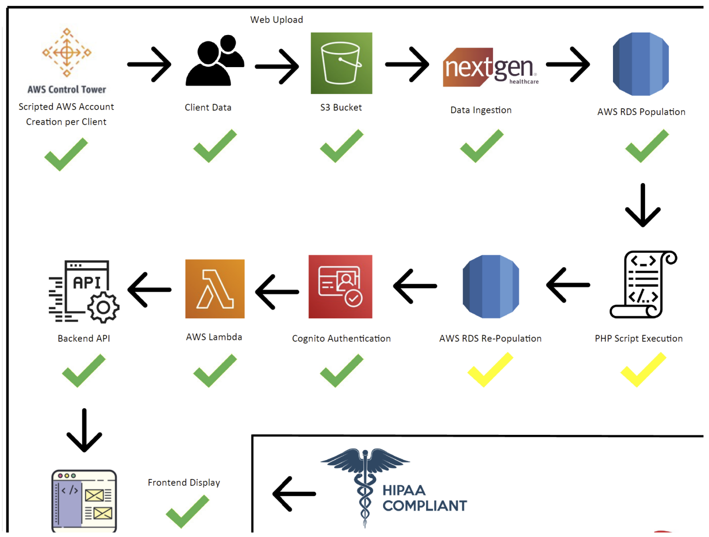
Use it to create something wonderful.

To start, you might want to:

- **Customise this overview** using the **edit icon** at the top right of this page.
- **Create a new page** by clicking the **+** in the space sidebar, then go ahead and fill it with plans, ideas, or anything else your heart desires.

Need inspiration?

- Get a quick intro into what spaces are, and how to best use them at Confluence 101: organize your work in spaces.
- Check out our guide for ideas on how to set up your space overview.
- If starting from a blank space is daunting, try using one of the space templates instead.

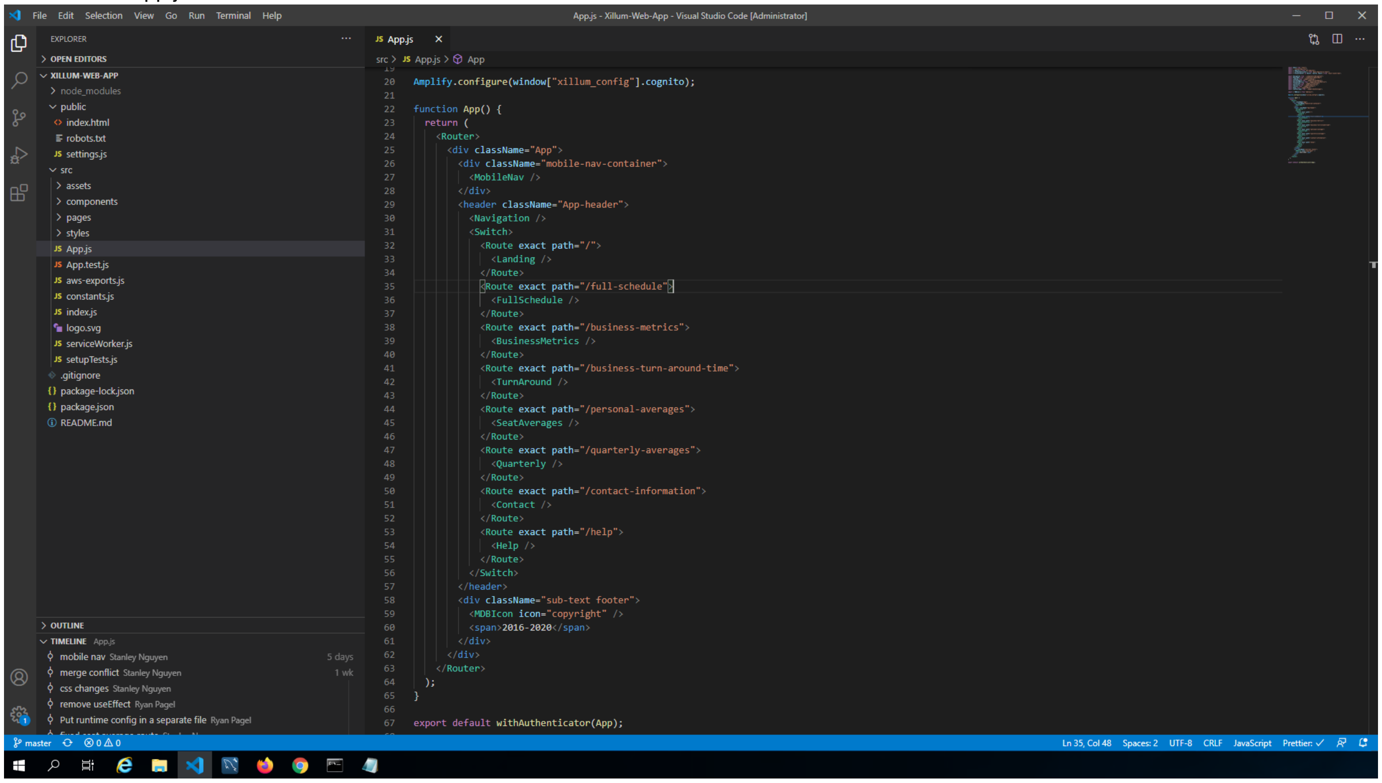


Frontend Overview

- Introduction:
- Navigation:
- Landing Page:
- Full Schedule:
- Business Metrics:
- Business Turn Around Time:
- Personal Seat Averages:
- Personal Quarterly Averages:
- Contact Information:
- Help:
- Other Helpful Information and Links:

Introduction:

The Xillum Billing Medical Dashboard is a React.js application styled with SASS, built with various libraries, connected to a Node.js backend application with mySQL serving as the database. AWS Cognito, built with AWS Amplify currently handles the authentication piece and will navigate the user through the login process. Should the user not be utilizing a proper JSON Web Token (JWT) upon signing in the user will be denied access to the site (Line 67 withAutheticator wraps App and utilized AWS Amplify to conduct authentication procedures). The Web Application currently has 8 independent pages that an authorized user can navigate to: Landing Page, Full Schedule , Business Metrics, Business Turn Around Time, Personal Seat Averages, Personal Quarterly Averages, Contact Information, and Help. Routing is done with React Router in the App.js file.



```
Amplify.configure(window["xillum_config"].cognito);

function App() {
  return (
    <Router>
      <div className="App">
        <div className="mobile-nav-container">
          <MobileNav />
        </div>
        <header className="App-header">
          <Navigation />
          <Switch>
            <Route exact path="/">
              <Landing />
            </Route>
            <Route exact path="/full-schedule">
              <FullSchedule />
            </Route>
            <Route exact path="/business-metrics">
              <BusinessMetrics />
            </Route>
            <Route exact path="/business-turn-around-time">
              <TurnAround />
            </Route>
            <Route exact path="/personal-averages">
              <SeatAverages />
            </Route>
            <Route exact path="/quarterly-averages">
              <Quarterly />
            </Route>
            <Route exact path="/contact-information">
              <Contact />
            </Route>
            <Route exact path="/help">
              <Help />
            </Route>
          </Switch>
        </header>
        <div className="sub-text footer">
          <MDBIcon icon="copyright" />
          <span>2016-2020</span>
        </div>
      </div>
    </Router>
  );
}

export default withAuthenticator(App);
```

Navigation:

React Router is responsible for the navigation of the application. Navigation.jsx in the components folder is where the web navigation can be found; whereas, MobileNav.jsx contains the mobile navigation component of the application. Both are documented here.

In Navigation.jsx, state is utilized to display the active tab in the navigation sidebar on line 20.

```
//Line 20

const [activeTab, setActiveTab] = useState();
```

handleActiveTab() is the function that takes in a single parameter (a number that designates which tab is clicked on) and sets the state

```
//Line 22

const handleActiveTab = (tabNum) => setActiveTab(tabNum);
```

In the Return statement is the .jsx that comprises the navigation.

MobileNav.jsx is the mobile navigation for the application and is a class-based component. It imports several items from “mdbreact” to create the component.

“State” holds the collapse ID of the link to toggle the opening and closing upon click.

toggleCollapse() is the function that takes in the collapse ID and compares it to the state to toggle the opening or closing.

```
//Line 23

toggleCollapse = (collapseID) => () => {
    this.setState((prevState) => ({
      collapseID: prevState.collapseID !== collapseID ? collapseID : "",
    }));
  };
```

The Return Statement holds the .jsx for the mobile nav.

Landing Page:

Landing.jsx is the component that comprises the main landing page. It can be found in the /src/pages folder. This page currently contains two tables: Today's Schedule and Your Upcoming Schedule.

"State" is utilized to hold data coming from the API for both tables.

```
//Line 7 and Line 8 respectfully

  const [todayData, setTodayData] = useState();
  const [upcomingData, setUpcomingData] = useState();
```

useEffect runs once upon the component mounts and fires the function: getData(). getData() does a GET call to the API with authorization headers to retrieve the required data to fill the tables. It is an asynchronous process. Once fetched, the data must be parsed into JSON. The date data is in UTC and must be converted to display properly in lines 24 to 31

```
//Lines 24 to 31

 fetchedRes.data.forEach((item) => {
        const date = new Date(item.date);
        const year = date.getUTCFullYear();
        const month = date.getUTCMonth();
        const day = date.getUTCDay();
        const todayDate = `${year}-${month}-${day}`;
        item.date = todayDate;
      });
```

At the end of the useEffect, the title for the browser tab is set on line 47.

```
//Line 47

document.title = "Xillum Radiologist Portal | Xillum";
```

The tables must be formatted in order to display them in the component. This is done in the constants data and upcomingTableData. The format consists of an object with a columns key (NOTE: the field key within the columns key reads the corresponding key in the data.) and rows key (state data is implemented here).

```
//Line 50-95

const data = {
    columns: [
      {
        label: "DATE",
        field: "date",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "RADIOLOGIST",
        field: "radiologist",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "SEAT",
        field: "site",
        sort: "asc",
        width: 150,
        height: 100,
      },
    ],
    rows: todayData,
  };

  const upcomingTableData = {
    columns: [
      {
        label: "DATE",
        field: "SchedDate",
        sort: "desc",
        width: 150,
        height: 100,
      },
      {
        label: "SEAT",
        field: "Seat",
        sort: "asc",
        width: 150,
        height: 100,
      },
    ],
    rows: upcomingData,
  };
```

The Return statement implements the jsx for the component with the formation of the tables. One thing to note is the <MDBDataTable /> utilizes different props to manipulate the table. More information on this component can be found here.

Full Schedule:

FullSchedule.jsx is the component that comprises of the Full Schedule page. It can be found in the /src/pages folder. This page currently contains two tables: Upcoming Schedule and Historic Schedule.

State is utilized to to hold data received from API after the GET call from getData in the useEffect. Additionally, the title of the page is updated in the useEffect.

```
//Lines 7-31

 const [scheduleData, setScheduleData] = useState();
 const [histData, setHistData] = useState();
  useEffect(() => {
    const getData = async () => {
      const path = `${window.xillum_config.app.hostname}/full-schedule
/upcoming-schedule`;
      const histPath = `${window.xillum_config.app.hostname}/full-
schedule/historic-schedule`;
      const webToken = (await Auth.currentSession())
        .getAccessToken()
        .getJwtToken();
      const myInit = {
        headers: {
          Authorization: `Bearer ${webToken}`,
        },
      };
      const fetched = await fetch(path, myInit);
      const fetchedRes = await fetched.json();
      setScheduleData(fetchedRes.schedules);

      const fetchedHist = await fetch(histPath, myInit);
      const fetchedHistRes = await fetchedHist.json();
      setHistData(fetchedHistRes.data);
    };
    getData();
    document.title = "Full Schedules | Xillum Radiologist Portal";
  }, []);
```

The tables must be formatted and is done so in the constants data and histTableData. The format consists of an object with a columns key (NOTE: the field key within the columns key reads the corresponding key in the data.) and rows key (state data is implemented here).

```
//Lines 33-187

const data = {
    columns: [
      {
        label: "Day",
          field: "daydate",
          sort: "asc",
          width: 200,
        },
        {
          label: "HIL",
          field: "HIL",
          sort: "asc",
          width: 200,
        },
        {
          label: "HPD",
          field: "HPD",
          sort: "asc",
          width: 250,
        },
        {
          label: "SVD",
          field: "SVD",
          sort: "asc",
          width: 150,
        },
        {
          label: "GVY",
          field: "GVY",
          sort: "asc",
          width: 150,
        },
        {
          label: "PNR",
          field: "PNR",
          sort: "asc",
          width: 150,
        },
        {
          label: "WDL",
          field: "WDL",
          sort: "asc",
          width: 150,
        },
        {
          label: "GWD",
          field: "GWD",
          sort: "asc",
          width: 150,
        },
        {
          label: "CLR",
          field: "CLR",
          sort: "asc",
        width: 150,
      },
      {
        label: "RSD",
        field: "RSD",
        sort: "asc",
        width: 150,
      },
      {
        label: "HZN/SAM",
        field: "HZN",
        sort: "asc",
        width: 150,
      },
      {
        label: "GFN/FFL",
        field: "GFN",
        sort: "asc",
        width: 150,
      },
    ],
    rows: scheduleData,
  };

  const histTableData = {
    columns: [
      {
        label: "Day",
        field: "daydate",
        sort: "asc",
        width: 150,
      },
      {
        label: "HIL",
        field: "HIL",
        sort: "asc",
        width: 150,
      },
      {
        label: "HPD",
        field: "HPD",
        sort: "asc",
        width: 150,
      },
      {
        label: "SVD",
        field: "SVD",
        sort: "asc",
        width: 150,
      },
        {
          label: "GVY",
          field: "GVY",
          sort: "asc",
          width: 150,
        },
        {
          label: "PNR",
          field: "PNR",
          sort: "asc",
          width: 150,
        },
        {
          label: "WDL",
          field: "WDL",
          sort: "asc",
          width: 150,
        },
        {
          label: "GWD",
          field: "GWD",
          sort: "asc",
          width: 150,
        },
        {
          label: "CLR",
          field: "CLR",
          sort: "asc",
          width: 150,
        },
        {
          label: "RSD",
          field: "RSD",
          sort: "asc",
          width: 150,
        },
        {
          label: "HZN/SAM",
          field: "HZN",
          sort: "asc",
          width: 150,
        },
        {
          label: "GFN/FFL",
          field: "GFN",
          sort: "asc",
          width: 150,
        },
      ],
      rows: histData,
    };
```

The Return statement implements the .jsx for the component with the formation of the tables. One thing to note is the <MDBDataTable /> utilizes different props to manipulate the table. More information on this component can be found here.

Business Metrics:

BusinessMetrics.jsx is the component that comprises of the Business Metrics page. It can be found in the /src/pages folder. This page currently contains two tables: wRVU by System and Total volume and Turn Around Time Detail. This page also contains six graphs: Modality Totals; ED and Inpatient Studies Completed by Hour on Each Day; Total Studies Completed by Hour on Each Day; Weekday Studies Completed and Read; Weekend ED CT TAT by Hour; and Emergency Department Turn Around Time by Modality

“State” is utilized to retain the data for use upon completion of the GET call done by the function getData() in the useEffect. Additionally, the title is changed on the component mounting.

```
//Line 8-45

 const [modTotalLabel, setModTotalLabel] = useState();
  const [modTotalData, setModTotalData] = useState();

  const [labelState, setLabelState] = useState();
  const [monData, setMonData] = useState();
  const [tuesData, setTuesData] = useState();
  const [wedData, setWedData] = useState();
  const [thurData, setThurData] = useState();
  const [friData, setFriData] = useState();
  const [satData, setSatData] = useState();
  const [sunData, setSunData] = useState();

  const [labelTotalState, setLabelTotalState] = useState();
  const [monTotalData, setMonTotalData] = useState();
  const [tuesTotalData, setTuesTotalData] = useState();
  const [wedTotalData, setWedTotalData] = useState();
  const [thurTotalData, setThurTotalData] = useState();
  const [friTotalData, setFriTotalData] = useState();
  const [satTotalData, setSatTotalData] = useState();
  const [sunTotalData, setSunTotalData] = useState();

  const [systemData, setSystemData] = useState();
  const [volumeData, setVolumeData] = useState();

  const [completeDateLabel, setCompleteDateLabel] = useState();
  const [completedData, setCompletedData] = useState();
  const [interpretedData, setInterpretedData] = useState();

  const [weekendLabel, setWeekendLabel] = useState();
  const [satWeekendData, setSatWeekendData] = useState();
  const [sunWeekendData, setSunWeekendData] = useState();

  const [edLabel, setEdLabel] = useState();
  const [crData, setCrData] = useState();
  const [ctData, setCtData] = useState();
  const [mrData, setMrData] = useState();
  const [usData, setUsData] = useState();
```

The useEffect fires getData once the component mounts to retrieve the data. Manipulation is done within getData for formatting date due to being originally in UTC.

```
//Lines 47-230
useEffect(() => {
    const getData = async () => ...
```

Lines 232-1008 is formatting for Chart.js which is used for visualizing the data into the bar chart and line charts.

Example from the code:

```
//Starting at Line 882

const volumeTableData = {
    columns: [
      {
        label: "DATE",
        field: "CompleteDate",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "ED COMPLETED",
        field: "ED Completed",
        sort: "asc",
        width: 150,
        height: 100,
      },
      ...
```

The charts take in the data and display in the Return statement within the components: <Bar /> and <Line />. To help configure these components. different props can be passed in; with “options“ to be the most noteworthy. It sets the labels and scales as seen below:

```
//Lines 951-970

 const totalOptions = {
    scales: {
      yAxes: [
        {
          scaleLabel: {
            display: true,
            labelString: "Study Count",
          },
        },
      ],
      xAxes: [
        {
          scaleLabel: {
            display: true,
            labelString: "Hour",
          },
        },
      ],
    },
  };
```

The Return statement implements the .jsx for the component with the formation of the tables. One thing to note is the <MDBDataTable /> utilizes different props to manipulate the table. More information on this component can be found here. Further documentation on the data visualizations can be found in Chart.js documentation found here at Chart.js. and the React part: here

Business Turn Around Time:

TurnAround.jsx is the component that comprises the Business Turn Around Time page. It can be found in the /src/pages folder. This page currently contains four tables: ED Turn Around Times; Inpatient Turn Around Times; Outpatient Turn Around Times; and ED Studies Falling Outside A 60 Minute Turn Around Time (30 day Summary).

“State” is utilized to retain the data for use upon completion of the GET call done by the function getData() in the useEffect. Additionally, the title is changed on the component mounting.

```
//Lines 6-9

  const [edTurnData, setEdTurnData] = useState();
  const [inPatientData, setInPatientData] = useState();
  const [outPatientData, setOutPatientData] = useState();
  const [outlierData, setOutlierData] = useState();
```

The useEffect fires getData() once the component mounts to retrieve the data.

```
//Lines 10-46

useEffect(() => {
    const getData = async () => ...
```

The tables required formatted data to display. Constants on Lines 48-133 hold the format.

```
Example within the code:

//Lines 115-133

const outlierTableData = {
    columns: [
      {
        label: "MODALITY",
        field: "Modality",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "NUMBER OF STUDIES",
        field: "NumberOfStudies",
        sort: "asc",
        width: 150,
        height: 100,
      },
    ],
    rows: outlierData,
  };
```

Once the tables are formatted, they can be used in the “data” prop for the component: <MDBDataTable /> in the Return statement. The different features of the table can be formatted through props passed into the component.

Personal Seat Averages:

SeatAverages.jsx is the component that comprises of the Personal Seat Averages page. It can be found in the /src/pages folder. This page currently contains two tables: Charged Personal Seat Averages and Your 30-Day Average.

“State” is utilized to retain the data for use upon completion of the GET call done by the function getData() in the useEffect and toggling the instructions. Additionally, the title is changed on the component mounting.

```
//Lines 7-10

  const [instructExpand, setInstructExpand] = useState(false);
  const [instructExpandTwo, setInstructExpandTwo] = useState(false);
  const [chargedData, setChargedData] = useState();
  const [daysData, setDaysData] = useState();
```

The useEffect fires getData() once the component mounts to retrieve the data.

```
//Lines 12-41

useEffect(() => {
    const getData = async () => ...
```

The tables required formatted data to display. Constants on Lines 43-101 hold the format.

```
Example below from the code.

//Lines 77-101

const daysTabledata = {
    columns: [
      {
        label: "SEAT",
        field: "Seat",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "RVU",
        field: "RVU",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "PROCEDURES",
        field: "Procedures",
        sort: "asc",
        width: 150,
        height: 100,
      },
    ],
    rows: daysData,
  };
```

The Return Statement displays the .jsx for the component. Once the tables are formatted, they can be used in the “data” prop for the component: <MDBDataTable />. The different features of the table can be formatted through props passed into the component.

Personal Quarterly Averages:

Quarterly.jsx is the component that comprises of the Personal Quarterly Averages page. It can be found in the /src/pages folder. The title is changed on the component mounting. This page currently contains one table and one bar chart for Your Quarterly Average.

“State” is utilized to retain the data for use upon completion of the GET call done by the function getData() in the useEffect. Additionally, the title is changed on the component mounting.

```
//Lines 9-12

  const [quarterData, setQuarterData] = useState();
  const [quarterLabel, setQuarterLabel] = useState();
  const [avgData, setAvgData] = useState();
  const [practiceData, setPracticeData] = useState();
```

The useEffect fires getData() once the component mounts to retrieve the data.

```
//Lines 13-43

useEffect(() => {
    const getData = async () => ...
```

The tables required formatted data to display. Constant on Lines 49-74 hold the format.

```
//Lines 49-74

const quarterTabledata = {
    columns: [
      {
        label: "QUARTER",
        field: "quarteryr",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "AVG DAILY RVU",
        field: "AvgDailyRvu",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "PRACTICE AVG RVU",
        field: "PracticeAvgRvu",
        sort: "asc",
        width: 150,
        height: 100,
      },
    ],
    rows: quarterData,
  };
```

For the barchart, the graph is formatted and the data is set in quarterBarData on line 76.

```
//Lines 76-90

const quarterBarData = {
    labels: quarterLabel,
    datasets: [
      {
        label: "Avg Daily RVU",
        backgroundColor: "#3281a8",
        data: avgData,
      },
      {
        label: "Practice Avg RVU",
        backgroundColor: "#b0191e",
        data: practiceData,
      },
    ],
  };
```

On line 92, the bar chart’s scales are set and formatted in the constant quarterlyOptions.

```
//Line 92

const quarterOptions = {
    scales: {
      yAxes: [
        {
          scaleLabel: {
            display: true,
            labelString: "RVU",
          },
        },
      ],
      xAxes: [
        {
          scaleLabel: {
            display: true,
            labelString: "Quarter",
          },
        },
      ],
    },
  };
```

The Return Statement displays the jsx for the component. Once the tables are formatted, they can be used in the “data” prop for the component: <MDBDataTable />. The different features of the table can be formatted through props passed into the component. <Bar /> is the component that displays the bar chart. The data prop passes through data to display into the bar chart and the options props sets the scales. Further documentation on the data visualizations can be found in Chart.js documentation found here at Chart.js. and the React part: here

Contact Information:

Contact.jsx is the component that comprises of the Contact Information page. It can be found in the /src/pages folder.

“State” is currently being used to toggle the instructions and hold data received from retrieving the data from the database.

```
//Line 10
const [instructExpand, setInstructExpand] = useState(false);
const [contactData, setContactData] = useState()
const [loading, setLoading] = useState(true);
```

The useEffect fires getData() once the component mounts to retrieve the data.

```
//Lines 14-33

useEffect(() => {
    const getData = async () => ...
```

The tables required formatted data to display. Constant on Lines 35-88 hold the format.

```
//Lines 35-88

const contactTableData = {
    columns: [
      {
        label: "SITE",
        field: "Site",
        sort: "asc",
        width: 150,
        height: 100,
      },
      {
        label: "TITLE",
        field: "Title",
        sort: "asc",
        width: 150,
        height: 100,
      },
      ...
```

The Return Statement displays the jsx for the component. Once the tables are formatted, they can be used in the “data” prop for the component: <MDBDataTable />. The different features of the table can be formatted through props passed into the component.

Help:

Help.jsx is the component that comprises of the Help page. It can be found in the /src/pages folder. The title is changed on the component mounting. The Accordion component is used from “react-bootstrap“ to display an accordion-style table.

```
//Lines 21-95 Part of the code to show how to add or change the table

<Accordion defaultActiveKey="0">
            <Card>
              <Card.Header>
                <Accordion.Toggle as={Button} variant="link" eventKey="
0">
                  <span>+</span>
                  <span className="help-header">Portal Description<
/span>
                </Accordion.Toggle>
              </Card.Header>
              <Accordion.Collapse eventKey="0">
                <Card.Body>
                  <PortalDescription />
                </Card.Body>
              </Accordion.Collapse>
            </Card>
            ...
```

More information can be found here.

Other Helpful Information and Links:

AWS Amplify for Authentication: https://docs.amplify.aws/lib/auth/getting-started/q/platform/js

Styling done in SASS (SCSS): https://sass-lang.com/documentation

React Router: https://reactrouter.com/web/guides/quick-start

Chart.js: https://www.chartjs.org/docs/latest/

React-Chartjs-2: https://github.com/jerairrest/react-chartjs-2

MDBReact: https://mdbootstrap.com/docs/react/

React Bootstrap: https://react-bootstrap.github.io/components/alerts/

- All Images should be stored in the /src/assets folder. Currently logo icon is stored here.

AWS Overview

- AWS SSO
- Updating products in the Xillum Service Catalog portfolio
- Cloud Trail Logs
- S3 Inventory
- New Customer Account Setup
    - Enroll a new customer account
    - Provision an enrolled customer account
    - Service Catalog
    - VPC
    - MySQL RDS Database
    - Cognito

    - DNS & SSL Certificate
    - Xillum REST API
    - Xillum Frontend
    - Secure S3 Upload Bucket
- To Be Deleted

AWS SSO

Overview

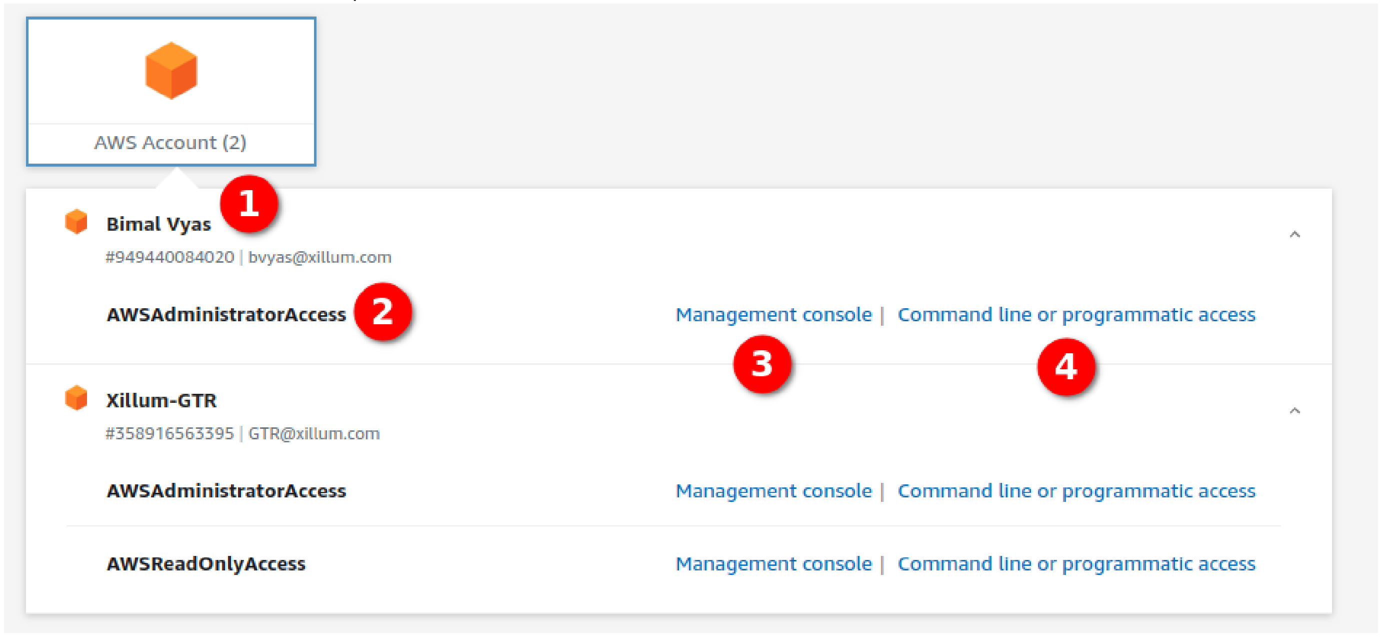
AWS SSO is automatically configured as part of the AWS Control Tower setup and is the recommended way to manage user access in a multi-account environment.

AWS SSO has the following advantages for vanialla IAM:

- Users only have to remember one set of credentials across many AWS accounts
- Provides a user portal for one stop account access. For Xillum the user portal URL is: https://d-906760816a.awsapps.com/start
- Centrally managed across the many accounts in an AWS organization
- Mult-factor auth can be easily be turned on and required for all users
- Allows for configuring with a external user directory like Active Directory

SSO User Portal

Here is an overview of what the user portal looks like:



Bimal Vyas
#949440084020 | bvyas@xillum.com
AWSAdministratorAccess — Management console | Command line or programmatic access

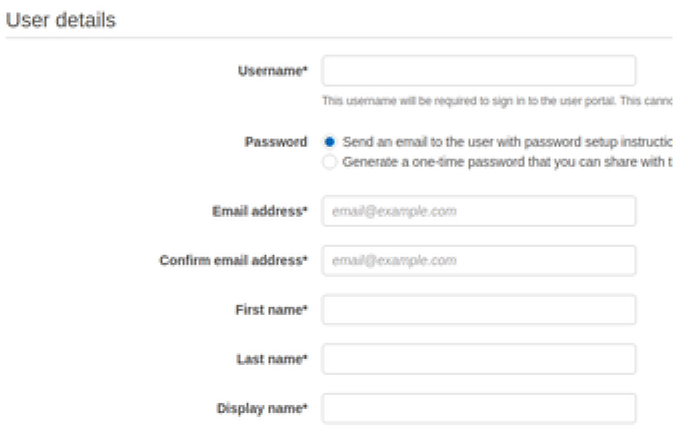
Xillum-GTR
#358916563395 | GTR@xillum.com
AWSAdministratorAccess — Management console | Command line or programmatic access
AWSReadOnlyAccess — Management console | Command line or programmatic access

1. Indicates the account name and account number
2. The list of assigned roles this user can use to login to the account
3. A link to login to the Management console
4. A link to generate command line/programmatic access credentials

Managing SSO Users

Head over to AWS SSO in the AWS console for the root account.

Click Users > Add user



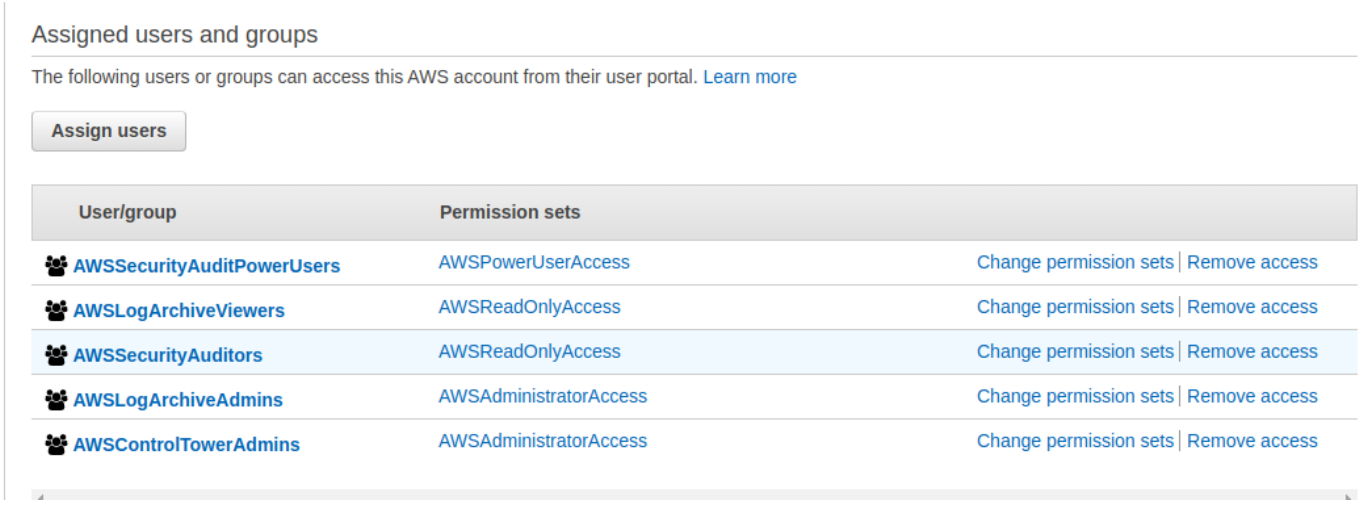
User details

Username*	
	This username will be required to sign in to the user portal. This cannc
Password	[✓ Send an email to the user with password setup instructic]
	Generate a one-time password that you can share with t
Email address*	email@example.com
Confirm email address*	email@example.com
First name*	
Last name*	
Display name*	

Fill in each field in the form. It’s easiest to allow the system to send an email directly to the user rather than having to send the email manually.

Optionally, add the user to an SSO group.

Users and groups can be assigned to various accounts and can be assigned **Permission sets** for those accounts.



Assigned users and groups

The following users or groups can access this AWS account from their user portal. Learn more

User/group	Permission sets	
AWSSecurityAuditPowerUsers	AWSPowerUserAccess	Change permission sets | Remove access
AWSLogArchiveViewers	AWSReadOnlyAccess	Change permission sets | Remove access
AWSSecurityAuditors	AWSReadOnlyAccess	Change permission sets | Remove access
AWSLogArchiveAdmins	AWSAdministratorAccess	Change permission sets | Remove access
AWSControlTowerAdmins	AWSAdministratorAccess	Change permission sets | Remove access

The Permission sets are the different “roles” the user can sign into the account as.

This setup allows for granularity in assigning permissions to various users and the accounts they can access.

Updating products in the Xillum Service Catalog portfolio

The `xillum-infra-pipeline` repository has the Infrastructure as Code (IaC) for all of the Service Catalog products except the REST API which is part of the `xillum-api-lambda` repository.

Generate the CloudFormation Template

The CloudFormation templates are created using the AWS CDK in Typescript. The process creating a new product version includes the following:

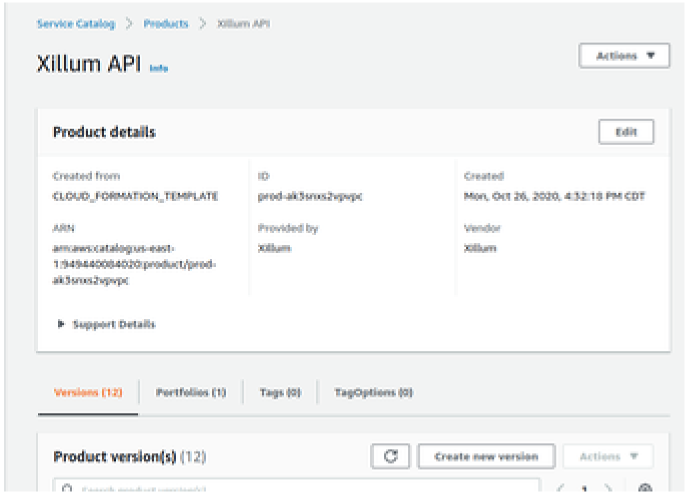
1. Download the source
2. `cd` into the **pipeline-cdk** directory
3. Run `npm install` to install all of the package dependencies necessary for the CDK.
4. Run the command `cdk synth [StackName] > [StackName].yml` . This will generate a template for the particular stack.
5. The different stacks correspond with the Service Catalog products and can be found in the `pipeline-cdk.ts` file and include:
    a. FrontendCdkStack02
    b. SecureS3CdkStack
    c. DbCdkStack
    d. VpcCdkStack
    e. CognitoCdkStack
    f. CertificateCdkStack

6. Copy the file into the appropriate location in the **xillum-service-catalog-templates** S3 bucket here. Each product has its own directory with a single `template.yml` file in it.

> The Xillum API product follows the same process but its IaC is located in the `xillum-api-lambda` repository. In the `infra` directory run the command `cdk synth XillumApiCdkStack02 > template.yml` to generate the API template.

Create the Service Catalog Product Version

Browse to Service Catalog > Administration > Products and click on the product you want to create a new version for. Click **Create new version**.


Service Catalog > Products > Xillum API

Xillum API

Product details

Created from	ID	Created
CLOUD_FORMATION_TEMPLATE	prod-ak3snxs2vpvpc	Mon, Oct 26, 2020, 4:32:18 PM CDT
ARN	Provided by	Vendor
arn:aws:catalog:us-east-1:949440084020:product/prod-ak3snxs2vpvpc	Xillum	Xillum

Support Details

Versions (12) | Portfolios (1) | Tags (0) | TagOptions (0)

Product version(s) (12)

On the Create screen:

- Select **Use a CloudFormation template**
- Enter the template URL using the https path to the file previously uploaded.
- Enter a unique version title (ideally an auto-incrementing to provide sorting capability).
- Enter a description of the reason for this version release.
- Click **Create product version**

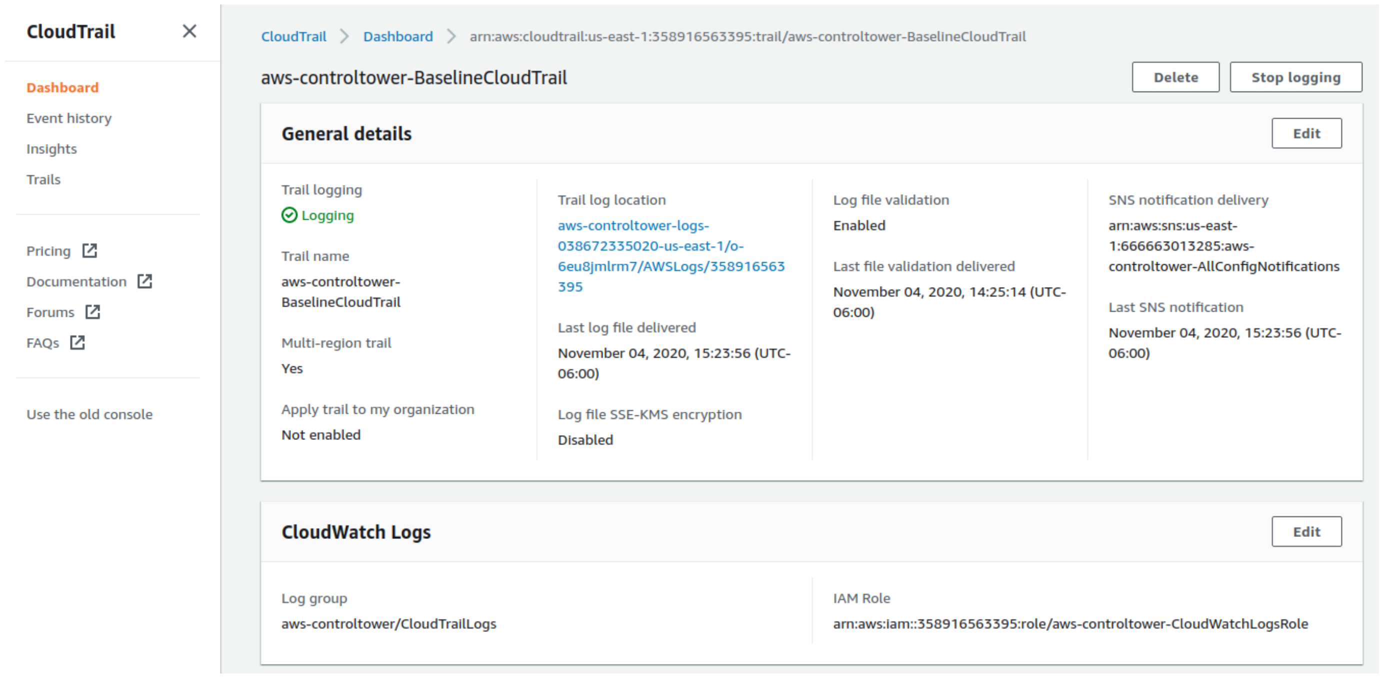
You have now create a new product version. This version will show up in the Service Catalog Products list for all accounts in the organization.

Cloud Trail Logs

By default AWS Control Tower turns on CloudTrail for every enrolled account. These logs are sent to two places:

- The central **Log Archive** account
- A CloudWatch log stream in the member account.

Head over to CloudTrail in a member account to see further details on how CloudTrail is configured.



aws-controltower-BaselineCloudTrail

General details

Trail logging
Logging

Trail name
aws-controltower-BaselineCloudTrail

Multi-region trail
Yes

Apply trail to my organization
Not enabled

Trail log location
aws-controltower-logs-038672335020-us-east-1/o-6eu8jmlrm7/AWSLogs/358916563395

Last log file delivered
November 04, 2020, 15:23:56 (UTC-06:00)

Log file SSE-KMS encryption
Disabled

Log file validation
Enabled

Last file validation delivered
November 04, 2020, 14:25:14 (UTC-06:00)

SNS notification delivery
arn:aws:sns:us-east-1:666663013285:aws-controltower-AllConfigNotifications

Last SNS notification
November 04, 2020, 15:23:56 (UTC-06:00)

CloudWatch Logs

Log group
aws-controltower/CloudTrailLogs

IAM Role
arn:aws:iam::358916563395:role/aws-controltower-CloudWatchLogsRole

S3 Inventory

The following is a listing of S3 buckets and their use. Blank comment sections indicate buckets that were not created by RSI.

Bucket Name	Comments
cdktoolkit-stagingbucket-1m0ev1o0iy4u9	Used by the CDK for development work in the master account. Created by running the `cdk bootstrap aws://949440084020/us-east-1` command.
elasticbeanstalk-us-east-1-949440084020	
elasticbeanstalk-us-east-2-949440084020	
cognitologineventpipeline-artifactsbucket2aac5544-14xiuabomqvug	Used by Codepipeline for storing the Cognito Login event Lambda build artifacts
frontendcdkstack-bucket83908e77-13h9u4fkdl14b	
frontendpipelinecdkstack-customerdeploymentbucket-16cqva06rqptq	Final deployment bucket where the Frontend compiled code is stored in versioned directories and reference from the customer accounts.
frontendpipelinecdkstack-frontendpipelineartifact-v611xzlauzp4	Used by Codepipeline for storing the frontend build artifacts
fwrintranetsupportfiles	
fwrradportal	
fwrradportal-200218	
pipelinecdkstack-expressapipipelineartifactsbucke-kc3hutx2qmvk	Used by Codepipeline for storing the artifacts for the first version of the REST API. Can be safely deleted.
xillum-api-bucket	Lambda code used
xillum-cognito-login-lambda	Lambda code used by the Cognito Login event handler deployed into customer accounts
xillum-secure-upload	Secure upload bucket used for GTR initially. This could be deleted.
xillum-service-catalog-templates	Service catalog templates used to deploy
xillumapipipelinecdkstac-frontendpipelineartifact-1pnndrittfj0x	Build artifacts bucket used by CodePipeline

New Customer Account Setup

This is a multi-step guide that walks you through how to enroll and provision a new customer account.

Enrollment

The first step is to create the new customer AWS account using Control Tower. Click Next at the bottom of this page or click here to go to the account enrollment documentation page.

Provisioning

After a new customer account has been enrolled, we can begin to provision that account. The account provisioning process is a multi-step process that has been automated as much as possible while allowing for flexibility and customization going forward.

We are using AWS Service Catalog to provision products into customer accounts. Service Catalog works seamlessly within AWS Organizations and uses CloudFormation under the covers, giving us the advantages of being able to define our infrastructure as code. Service Catalog also uses a versioning system where new product version can be rolled out selectively to different customers.

The following is an overview of the account provisioning steps:

- Setup Service Catalog access
- VPC
- MySQL RDS Database
- Cognito
- DNS and SSL Certificate
- REST API
- Frontend
- Secure S3 Upload Bucket

Most of these steps are point-and-click but there are some manual steps included and we will cover all of it in this guide.

Each product provisioning page in this documentation will include at least the following sections:

- **Provision**: how to provision the product
- **Details**: what exactly is created by the product
- **Outputs**: variable values that have been stored in either Secrets Manager or Parameter Store to be reference by other products. This reduces the amount of value copy/pasting required by the person provisioning these products.
- **Next**: the next step in the multi-step process.

Some pages will include additional information such as RDS where we cover disaster recovery considerations.

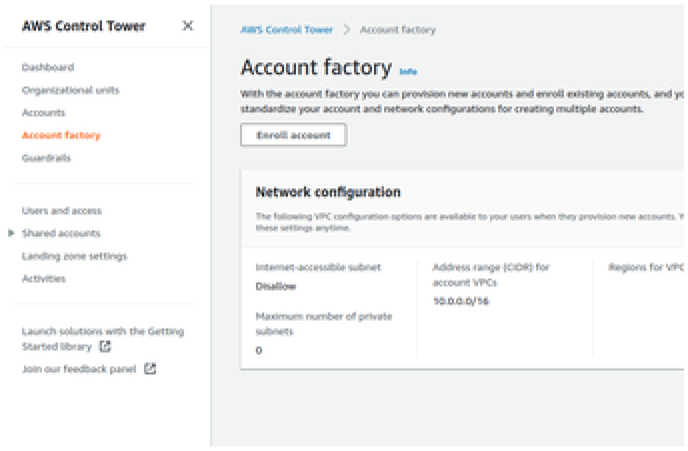
Next

Begin by enrolling the new customer account

Next: Account Enrollment

Enroll a new customer account

Adding a new customer is a process we have thought through and attempted to make as streamlined as possible while allowing for flexibility and customization going forward. Head over to AWS Control Tower Dashboard in the primary AWS account to begin the process. From the AWS Control Dashboard, click on Account Factory on the left. Do not make any changes to the Network Configuration, then click Enroll Account.


AWS Control Tower

Dashboard
Organizational units
Accounts
Account factory
Guardrails

Users and access
Shared accounts
Landing zone settings
Activities

Launch solutions with the Getting Started library
Join our feedback panel

AWS Control Tower > Account factory

Account factory Info

With the account factory you can provision new accounts and enroll existing accounts, and yo
standardize your account and network configurations for creating multiple accounts.

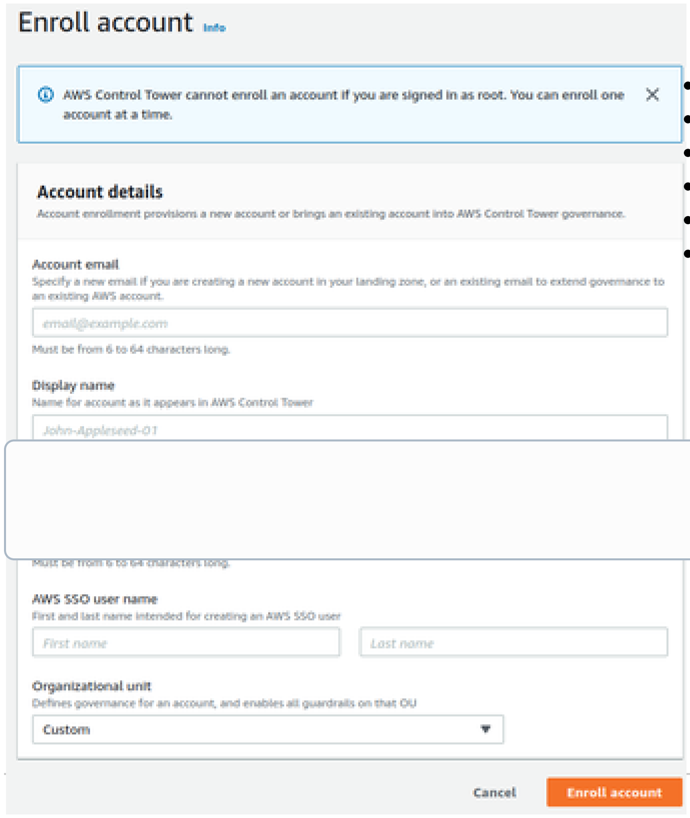
Enroll account

Network configuration

The following VPC configuration options are available to your users when they provision new accounts. Y
these settings anytime.

Internet-accessible subnet: Disallow
Maximum number of private subnets: 0
Address range (CIDR) for account VPCs: 10.0.0.0/16
Regions for VPC

You will see the Enroll Account screen like below.


Enroll account Info

AWS Control Tower cannot enroll an account if you are signed in as root. You can enroll one account at a time.

Account details

Account enrollment provisions a new account or brings an existing account into AWS Control Tower governance.

Account email
Specify a new email if you are creating a new account in your landing zone, or an existing email to extend governance to an existing AWS account.
email@example.com
Must be from 6 to 64 characters long.

Display name
Name for account as it appears in AWS Control Tower
John-Appleseed-01

Must be from 6 to 64 characters long.

AWS SSO user name
First and last name intended for creating an AWS SSO user
First name | Last name

Organizational unit
Defines governance for an account, and enables all guardrails on that OU
Custom

Cancel | Enroll account

Enter the following:

- Account email (must be a unique, valid email address)
- Account Display Name
- AWS SSO email
- AWS SSO user first name
- AWS SSO user last name
- Leave the Organizational Unit as Custom

Click Enroll Account.

It will take some time for the account to be provisioned. Once it is you can login to the account using AWS SSO. See the SSO page for more on AWS SSO and how to use it with Xillum.

Control Tower requires adding an SSO user to the account during initial account enrollment. This user can be deleted later if it is not required.

Next

Once an account has been enrolled through Control Tower we can begin to provision that account. Provisioning, for our use, is a process of installing Service Catalog products in the account in order to run the Xillum application in the account.

Next: Setup Service Catalog

Provision an enrolled customer account

After a new customer account has been enrolled, we can begin to provision that account. The account provisioning process is a multi-step process that has been automated as much as possible while allowing for flexibility and customization going forward.

The following is an overview of the account provisioning steps:

- Setup Service Catalog access
- Networking
- DNS and certificate
- RDS Database
- Cognito (user management)
- Secure S3 upload bucket
- Xillum REST API
- Xillum Frontend

Most of these steps are point-and-click but there are some manual steps included and we will cover all of it in this guide.

Each product provisioning page in this documentation will include at least the following sections:

- **Provision**: how to provision the product
- **Details**: what exactly is created by the product
- **Outputs**: variable values that have been stored in either Secrets Manager or Parameter Store to be reference by other products. This reduces the amount of value copy/pasting required by the person provisioning these products.
- **Next**: the next step in the multi-step process.

Some pages will include additional information such as RDS where we cover disaster recovery considerations.

Next

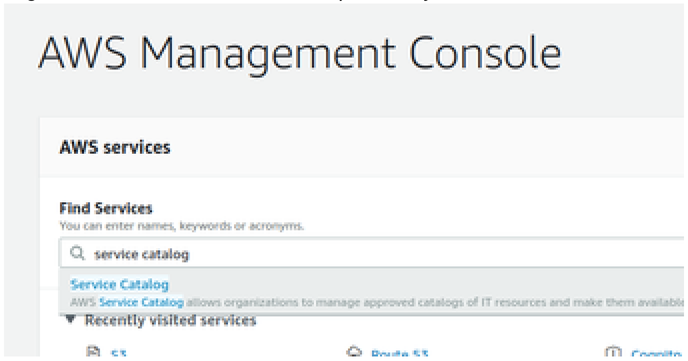
Begin provisioning the customer account by setting up Service Catalog

---

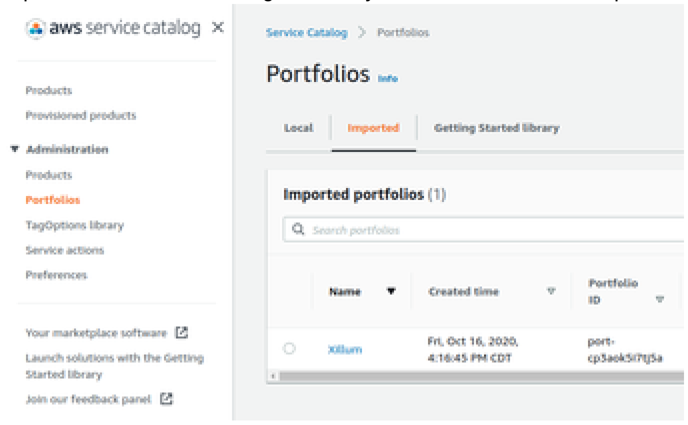
Next: Setup Service Catalog

Service Catalog

Login to the new customer account previously enrolled in Control Tower and head over to Service Catalog.


AWS Management Console
AWS services
Find Services
You can enter names, keywords or acronyms.
service catalog
Service Catalog
AWS Service Catalog allows organizations to manage approved catalogs of IT resources and make them availabl
Recently visited services
S3
Route 53
Cognito

In Service Catalog, click Portfolios under the Administration section in the left-hand navigation. Then click on the Imported tab. Since this account is part of the Xillum AWS organization you should see the Xillum portfolio listed under Imported Portfolios. Click on the Xillum portfolio.



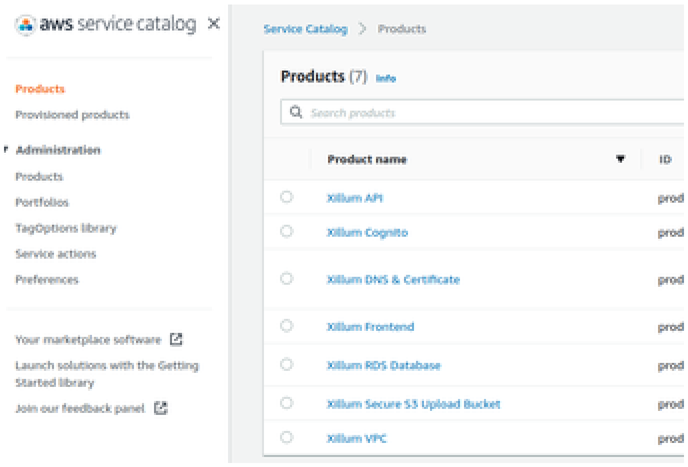
	Name	Created time	Portfolio ID
○	Xillum	Fri, Oct 16, 2020, 4:16:45 PM CDT	port-cp3aok5i7tj5a

Next click on Groups, roles, and users > Add groups, roles, users. Click on the Roles tab and add the following roles:

- AWSReservedSSO_AWSAdministratorAccess
- AWSReservedSSO_AWSPowerUserAccess

Click **Add access**.

Once the access has been added you should see the products that are part of the **Xillum** portfolio listed under Products in the left navigation. Click here to go directly to that page. You should see the products listed that are available to be provisioned into this account. The **Products** page lists the products that are available to be provisioned into this account while the **Provisioned products** page lists the products that have actually been provisioned into this account.



Product name	ID
Xillum API	prod
Xillum Cognito	prod
Xillum DNS & Certificate	prod
Xillum Frontend	prod
Xillum RDS Database	prod
Xillum Secure S3 Upload Bucket	prod
Xillum VPC	prod

Next

Once the Xillum Service Catalog portfolio is setup you can provision the first product, which is the VPC.

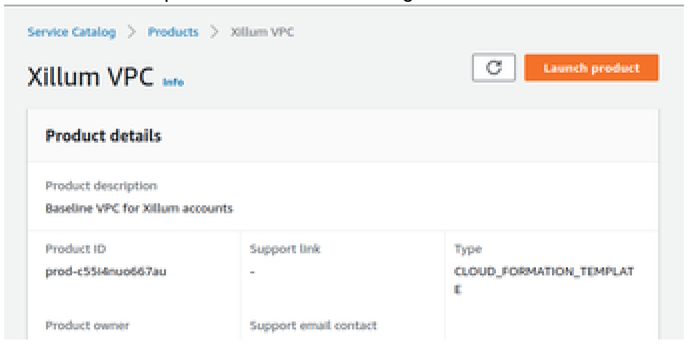
Next: VPC »

VPC

Provisioning the VPC

The first step to setting up a new customer's AWS account is to provision a VPC. Although Control Tower comes with a VPC we have disabled the creation of a VPC upon enrolling a new account through Control Tower. This allows us to fully control the VPC creation as well as potentially roll out updates more easily in the future.

To provision the account's VPC browse to AWS Service Catalog. Click on Products, then click Xillum VPC from the product list. This will display details about the product in Service Catalog as well as the versions that are available to be deployed. Click **Launch Product**.



Product details

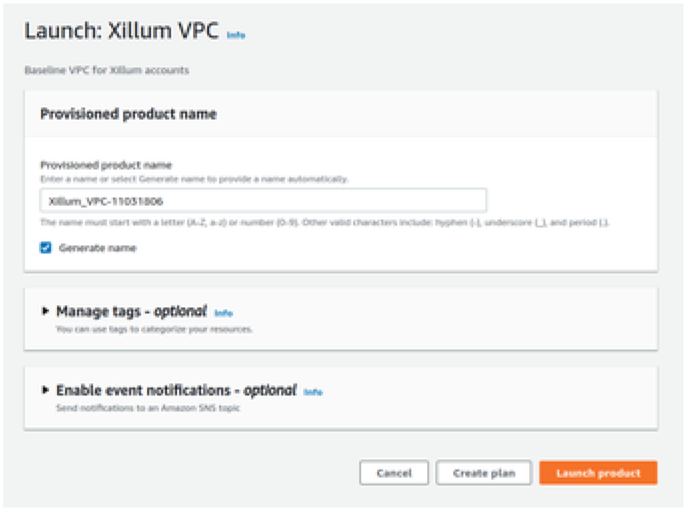
Product description
Baseline VPC for Xillum accounts

Product ID	Support link	Type
prod-c55l4nuo667au	-	CLOUD_FORMATION_TEMPLATE

Product owner
Support email contact

here we can enter the product name (one is provided by default). Remove the trailing numbers from the product name. Next, select a version to be deployed. In all cases we will want to select the most recent version which should also be the highest version number. `1.0.10` is a higher version number than `1.0.6`. In the case of the Xillum VPC there is only one version number so there is no selection box for the version. Most of the other products will have multiple versions, just make sure to choose the most recent one. Click **Launch product** again.

> Please be aware that other products will have additional parameters to modify on this second Launch screen. The VPC product does not have any but that is not the case with most other products.


Launch: Xillum VPC Info

Baseline VPC for Xillum accounts

Provisioned product name

Provisioned product name
Enter a name or select Generate name to provide a name automatically.

Xillum_VPC-11031806

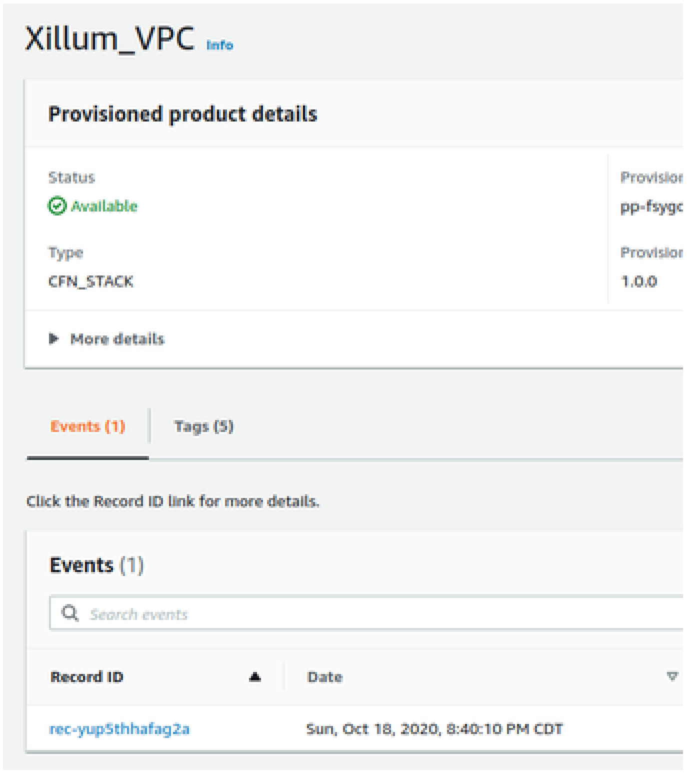
The name must start with a letter (A-Z, a-z) or number (0-9). Other valid characters include: hyphen (-), underscore (_), and period (.).

[✓] Generate name

▸ **Manage tags - *optional*** Info
You can use tags to categorize your resources.

▸ **Enable event notifications - *optional*** Info
Send notifications to an Amazon SNS topic

To view the progress of provisioning progress you can wait on this screen until it completes. If there is an error provisioning the product it will also appear here. To view more detailed provisioning status you can view the CloudFormation details by clicking the most recent Record ID in the Events section.


Xillum_VPC Info

Provisioned product details

Status	Provisior
✓ Available	pp-fsygc
Type	Provisior
CFN_STACK	1.0.0

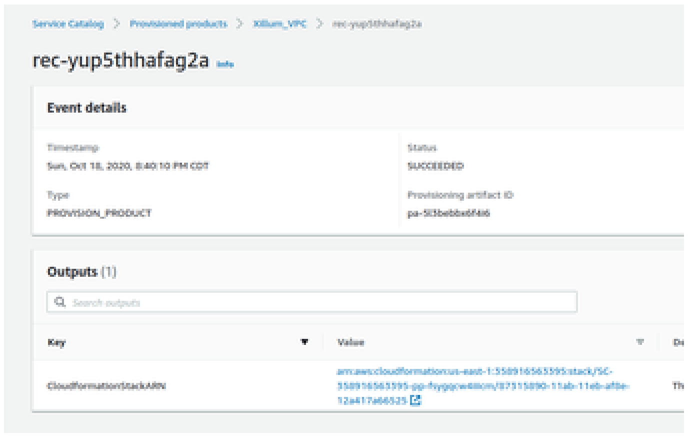
▸ More details

Click the Record ID link for more details.

Events (1)

Record ID	Date
rec-yup5thhafag2a	Sun, Oct 18, 2020, 8:40:10 PM CDT

From here you can click on the link to the CloudFormation stack.


Service Catalog > Provisioned products > Xillum_VPC > rec-yup5thhafag2a

rec-yup5thhafag2a Info

Event details

Timestamp	Status
Sun, Oct 18, 2020, 8:40:10 PM CDT	SUCCEEDED
Type	Provisioning artifact ID
PROVISION_PRODUCT	pa-5l3bebbx6f4i6

Outputs (1)

Key	Value	De
CloudformationStackARN	arn:aws:cloudformation:us-east-1:550916563595:stack/SC-550916563595-pp-fsygg[illegible]/87315890-11ab-11eb-af8e-12a417a66525	Th

On the CloudFormation screen you can view all of the details related to this product, such as the raw CFN template used, the individual resources created and a list of events happening in real-time.

VPC Details

The VPC created will include the following:

- Once VPC with a 10.0.0.0/16 CIDR range.
- Two public subnets spanning two Availability Zones
- Two private subnets spanning the same two Availability Zones
- Two NAT Gateways (one in each public subnet) so the private subnets can have outbound internet access. The private subnets will each have a route to their respective NAT Gateway.
- VPC Flow logs turned on and sent to a CloudWatch log group

Outputs

The Xillum VPC product will output a couple of values, stored as Parameter Store parameters and can be viewed in the customer account here. The parameters are:

Parameter Name	Description
/vpc/id	The VPC Id
/vpc/private_subnets	A comma-delimited list of the VPC's private subnets

These outputs will be used by some of the upcoming products we will provision.

Next

Once our VPC is setup we can proceed to provision our RDS MySQL Database.

Next: MySQL RDS Database »

MySQL RDS Database

Provision

To provision the MySQL RDS Database, navigate to the Products screen in the AWS Service Catalog. From the list of Products, select Xillum RDS Database, select Launch product, adjust the Product Name as preferred, select the most recent version and click Launch product.

It may take 10-15 minutes for the RDS Database to be deployed. Provisioning details can be viewed in CloudFormation as described in the VPC documentation.

Copy the password from the Secrets Manager and create a new SSM parameter called `/dev/mysql_password`. We didn't have time to re-reference the password in Secrets Manager from the code but this could be done later. Follow these steps:

- Head over to Secrets Manager and click on `xillum-rds-password`.
- Scroll down and click `Retrieve secret value`.

- The username and password will be listed. Make note of the username and copy the password.
- Head over to Parameter Store here.
- Create a `Secure String` parameter named `/dev/mysql_password`. Paste the value copied from Secrets Manager.

You have not added the DB password to Parameter Store. This will allow future products we will provision to reference the db password.

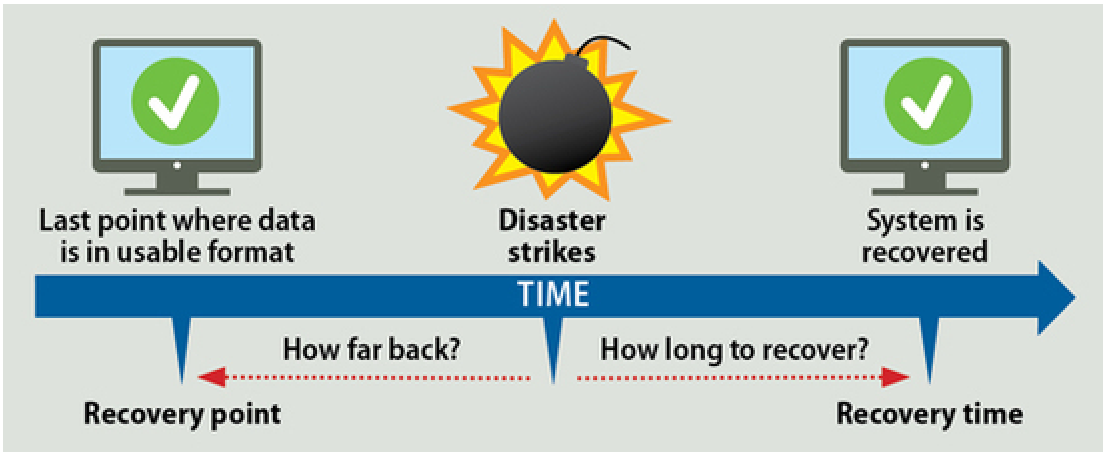
Details

This product will create a single instance MySQL RDS database and can be viewed at the RDS instances page in the AWS console.

- db.t3.medium (2 vCPU, 4 GB RAM).
- MySQL 8.0.20
- No public access. The database instance runs in the private subnets and can only be access from within the VPC.
- Encryption at rest is enabled using the default KMS key (aws/rds).
- Automated backups turned on which enables Point-in-time-recovery (PITR).
- Autoscaling storage from 100GB up to 1TB.
- The `master` user credentials are created and stored in Secrets Manager as described above.

Disaster Recovery Considerations

Disaster recovery is expressed in terms of RTO (Recovery Time Objective) and RPO (Recovery Point Objective). The RTO is how long it takes to recover a system back to a good state, while RPO signifies how far back we must go in time to get a good backup. Some applications can afford to go back further in time for its last backup while others cannot. Other applications cannot afford to have any downtime while some application can afford several hours of downtime. These options must always be considered in relation to cost since decreasing RTO will often mean significant additional cost.



Credit: https://www.enterprisestorageforum.com/storage-management/rpo-and-rto-understanding-the-differences.html

For the purposes of Xillum we have chosen a single MySQL RDS instance with automated point-in-time backups enabled. This provides for a RPO of nearly 0 since we can restore from any point in time. The RTO would be however long it takes to restore a new RDS instance from a PIT snapshot and point the application at that instance. This could be accomplished in a at most a few hours. We feel that this choice is a good balance of disaster recovery performance and cost. If there is a desire to further reduce RTO, RDS provides other options but at significant cost increases.

- **Database clustering.** RDS Aurora provides full db clustering support with read replication and multi-master options as described here. We are not using RDS Aurora but rather MySQL so the Aurora clustering capabilities are currently not available to us.
- **Multi-AZ**. This option provides an additional RDS instance in the VPC that lives in another Availability Zone. RDS multi-AZ functions as a warm failover standby and automatically fails over to the secondary instance in case of a disaster scenario. This can help reduce RTO time.
- **Read replica**. A read replica can further bolster application read performance while also allowing the replica to be manually promoted in disaster recovery scenarios.
- **Combination of read replica and multi-AZ.** Multi-AZ and Read Replica can also be combined for further application and RTO performance.

See the AWS documentation for more on the differences between read replicas and multi-AZ: https://aws.amazon.com/rds/features/read-replicas/

Outputs

- Secrets Manager credentials

- /db/hostname in Parameter Store

Next

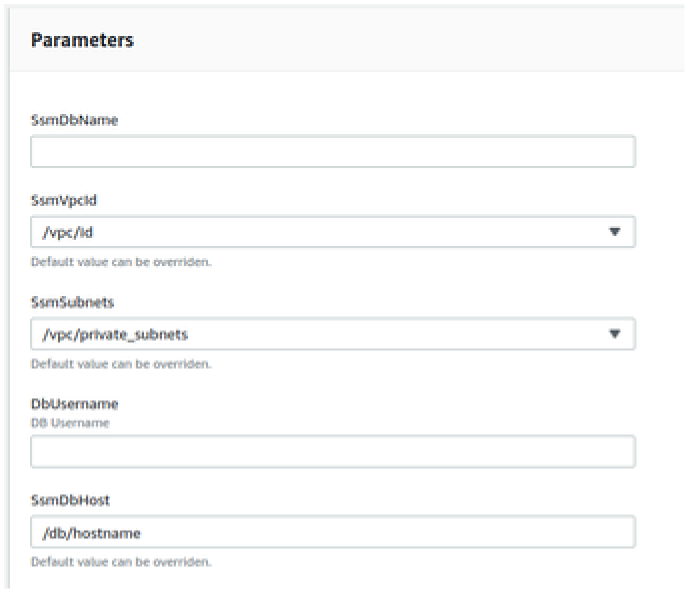
Once the MySQL RDS Database is provisioned we can provision Cognito since the Cognito product has a dependency on the database (this will be explained in the Cognito documentation).

Next: Cognito >>

Cognito

Provision

To provision the Cognito product head over to AWS Service Catalog, click on Product and select Xillum Cognito. Launch the product using the latest version and fill in the required parameters. The `SsmDbName` should be `xillum` and the `DbUsername` should be the value copied in the RDS provisioning step which is `master` by default. The other parameters will be pulled from Parameter Store automatically and shouldn't be modified.



Parameters

SsmDbName

SsmVpcId
/vpc/id
Default value can be overriden.

SsmSubnets
/vpc/private_subnets
Default value can be overriden.

DbUsername
DB Username

SsmDbHost
/db/hostname
Default value can be overriden.

Required Manual Step. In order to complete the Cognito provisioning we need to add the public key for our Cognito user pool to the Parameter Store. This is described here in the AWS documentation for reference. Take the following URL as a template and substitute `us-east-1` for the region and grab the userPoolId value from Parameter Store item with a key of `/cognito/user_pool_id`.

https://cognito-idp.{region}.amazonaws.com/{userPoolId}/.well-known/jwks.json

After substituting the values we will have a URL that looks like this:

https://cognito-idp.us-east-1.amazonaws.com/us-east-1_2zCZ7h9fu/.well-known/jwks.json

Browse to the above URL in a web browser and copy the entire value exactly as it appears. Create a new Parameter Store item called `/cognito/keys` and paste that value in there.

Details

Creates the following:

- Cognito Identity pool
- Cognito User pool
- Congito app client
- Cognito login event handler Lambda function: adds new users to the `cognito_users` table.

Outputs

- /cognito/app_client_id
- /cognito/user_pool_id

- /cognito/identity_pool_id
- /cognito/keys: this value must be added manually as described above in the Provision step.

Next

The DNS and DNS certificate can be provisioned next.

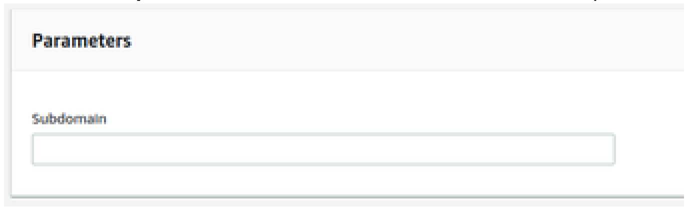
Next: DNS & Certificate

DNS & SSL Certificate

Provision

Browse to AWS Service Catalog in the customer account, select Products, then select Xillum DNS & Certificate.

Click **Launch product**, select the most recent **version**, then input the **subdomain** parameter as shown below.



The value entered for `Subdomain` should only be the subdomain, not the entire domain. For example, enter `gtr` rather than `gtr.myxillum.com`.

Click Launch product again to initiate the provisioning.

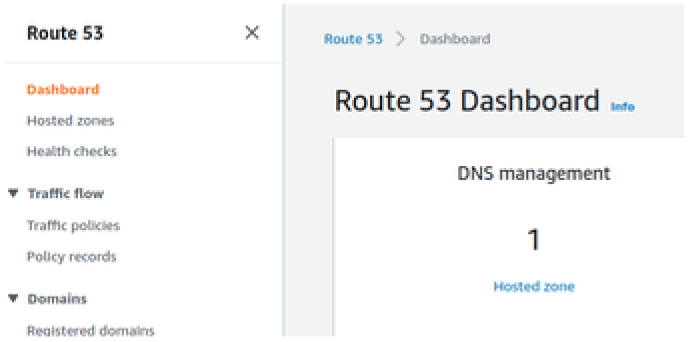
> Note, this will create an SSL certificate for gtr.myxillum.com and *.gtr.myxillum.com. The provisioning process will not complete until AWS Certificate Manager can verify that you own this subdomain.

The following are the manual steps that will have to be completed to finish the provisioning process, allowing AWS Certificate Manager to verify your domain ownership.

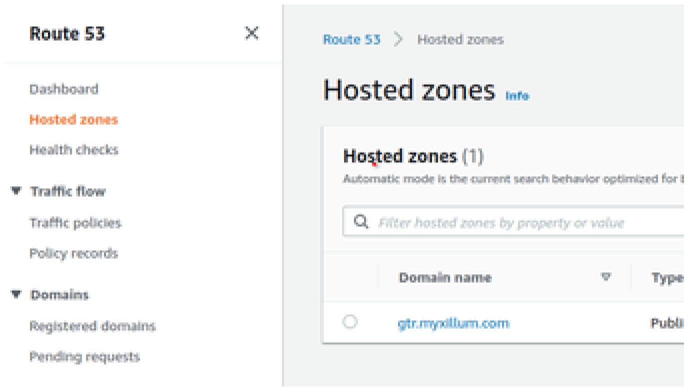
Manual Steps

For the purposes of documentation we'll assume that the subdomain we're working with is `gtr.myxillum.com`. For future accounts substitute your new subdomain for `gtr`.

Grab the hosted zone NS record value from Route 53. Head over to Route 53, selected Hosted Zone.



Select the domain name for your newly created Hosted Zone (as part of this product).



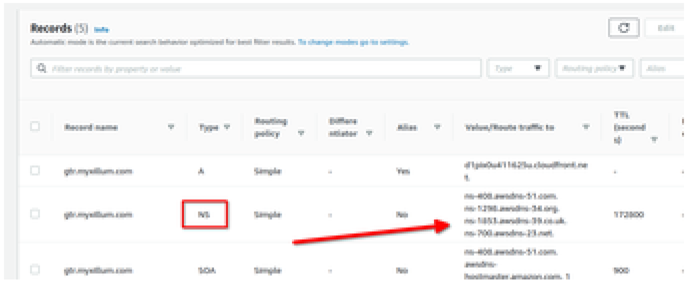
Hosted zones Info

Hosted zones (1)

Automatic mode is the current search behavior optimized for

Domain name	Type
gtr.myxillum.com	Publi

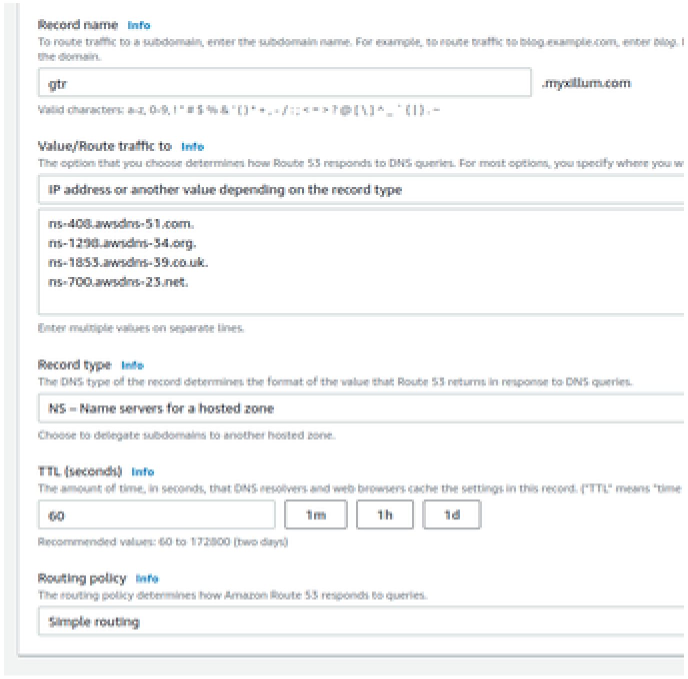
Copy the value for the NS record for `gtr.myxillum.com`.


Records (5) Info

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

Record name	Type	Routing policy	Differentiator	Alias	Value/Route traffic to	TTL (seconds)
gtr.myxillum.com	A	Simple	-	Yes	d1ple0u411623u.cloudfront.net.	-
gtr.myxillum.com	NS	Simple	-	No	ns-408.awsdns-51.com. ns-1298.awsdns-34.org. ns-1853.awsdns-39.co.uk. ns-700.awsdns-23.net.	172800
gtr.myxillum.com	SOA	Simple	-	No	ns-408.awsdns-51.com. awsdns-hostmaster.amazon.com. 1	900

Now, login to your root AWS Organization account and head over to the Route 53 console. This account is the owner of the `myxillum.com` domain. We're going to delegate management of the `gtr.myxillum.com` subdomain to our newly created hosted zone in the new customer account. To do this add an NS record for `gtr.myxillum.com` pasting in the NS records you just copied as the value:


Record name Info

To route traffic to a subdomain, enter the subdomain name. For example, to route traffic to blog.example.com, enter blog. I the domain.

gtr .myxillum.com

Valid characters: a-z, 0-9, ! " # $ % & ' ( ) * + , - / : ; < = > ? @ [\ ] ^ _ ` { | } . ~

Value/Route traffic to Info

The option that you choose determines how Route 53 responds to DNS queries. For most options, you specify where you w

IP address or another value depending on the record type

```
ns-408.awsdns-51.com.
ns-1298.awsdns-34.org.
ns-1853.awsdns-39.co.uk.
ns-700.awsdns-23.net.
```

Enter multiple values on separate lines.

Record type Info

The DNS type of the record determines the format of the value that Route 53 returns in response to DNS queries.

NS – Name servers for a hosted zone

Choose to delegate subdomains to another hosted zone.

TTL (seconds) Info

The amount of time, in seconds, that DNS resolvers and web browsers cache the settings in this record. ("TTL" means "time

60 1m 1h 1d

Recommended values: 60 to 172800 (two days)

Routing policy Info

The routing policy determines how Amazon Route 53 responds to queries.

Simple routing

This will now send all DNS lookup requests for `gtr.myxillum.com` to the new account. If this was performed correctly your product should finish provisioning since AWS Certificate Manager can now verify that you own this subdomain. The DNS validation records are added automatically by the Service Catalog product to the Hosted Zone in the new account. Navigate over to the Route 53 console in the new account to view what it looks like.

Details

Creates the following:

- An SSL certificate for `gtr.myxillum.com` and `*.gtr.myxillum.com`.
- A Route 53 DNS Hosted Zone for `gtr.myxillum.com`. After the DNS is delegated to this account (as described above), all DNS for the customer is managed in their own account.

Outputs

- /acm/cert_arn
- /dns/hosted_zone_id

Next

Next we can deploy the Xilllum REST API.

Next: Xillum REST API

Xillum REST API

Provision

Head over to the AWS Service Catalog in the customer account, click on Products and then click on the Xillum API product. Click Launch product, select and version and enter the required Parameters. The following Parameters need to be manually entered:

- SsmDbName: `xillum`
- DbUsername: `master`

All of the other values are pulled from prior Parameter Store values.

Click Launch product and wait for it to finish.

Details

The following AWS resources are created by this product:

- A lambda function that runs our serverless NodeJs API. This lambda function pulls its source code from a cross-account S3 bucket that is hosted in the root Organization account.
- An API Gateway that has the following features enabled:
    - JWT authorization to verify the JWT tokens sent from the frontend
    - CORS to prevent cross-origin requests for any domains other than `api.gtr.myxillum.com`.
    - The API Gateway proxies requests to the Lambda function
- A custom domain name mapping `api.gtr.myxillum.com` to this API Gateway.
- A DNS entry in Route 53 pointing `api.gtr.myxillum.com` to the DNS name for the custom domain.

Outputs

There are not Parameter Store outputs for this product.

Next

Next, we can provision the Xillum Frontend pieces.

Next: Xillum Frontend

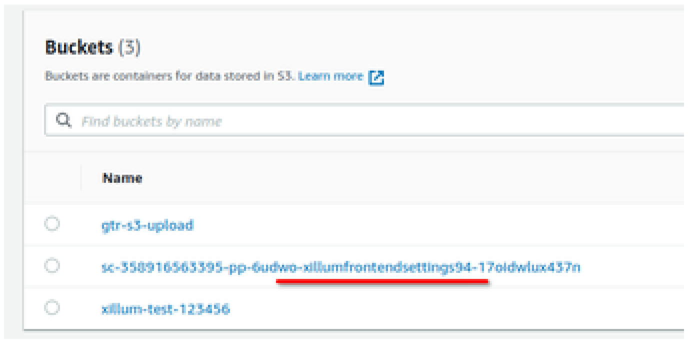
Xillum Frontend

Provision

In the Service Catalog products list, click **Launch Product** on `Xillum Frontend`. There are no Parameters to enter. Once the provisioning completes, the following manual steps will need to be completed.

Frontend Configuration

After the provisioning of this stack completes, you will see an S3 bucket that has the word `xillumfrontendsettings` in it (see the below image).


Buckets (3)

Buckets are containers for data stored in S3. Learn more

gtr-s3-upload

sc-358916563395-pp-6udwo-xillumfrontendsettings94-17oldwlux437n

xillum-test-123456

We need to add a file to this bucket with the name `settings.js`. The values that are in all caps need to be replaced with their appropriate values from Parameter Store here. Here is the `settings.js` file template.

```
window.xillum_config = {
    cognito: {
        "aws_project_region": "us-east-1",
        "aws_cognito_identity_pool_id": "/COGNITO/IDENTITY_POOL_ID",
        "aws_cognito_region": "us-east-1",
        "aws_user_pools_id": "/COGNITO/USER_POOL_ID",
        "aws_user_pools_web_client_id": "/COGNITO/APP_CLIENT_ID",
        "oauth": {}
    },
    "app": {
        "hostname": "https://api.SUBDOMAIN.myxillum.com"
    }
}
```

Find the following values in the template and replace them with the corresponding values from Parameter Store that match these keys:

- `/COGNITO/IDENTITY_POOL_ID`
- `/COGNITO/USER_POOL_ID`
- `/COGNITO/APP_CLIENT_ID`
- `SUBDOMAIN` replace with `/dns/subdomain`from Parameter Store.

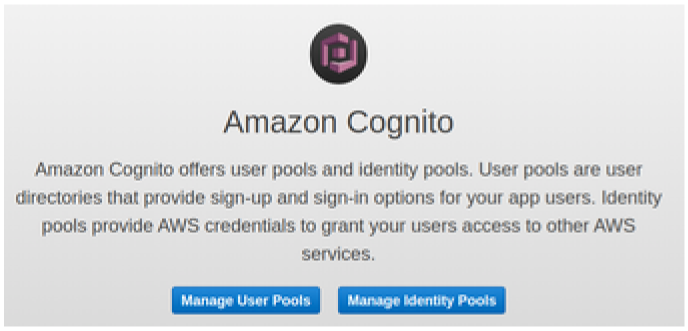
Upload this new file to the S3 bucket with the name `settings.js`.

> Moving the application settings out into a separate file allows us to use the same Frontend codebase for multiple customers. By not baking these settings into the compiled Frontend code we can use the same Frontend built code for every customer by performing the above configuration.

Cognito User Creation

Now that we've provisioned the Frontend we can actually login to the website. First we need to create a Cognito user to login with.

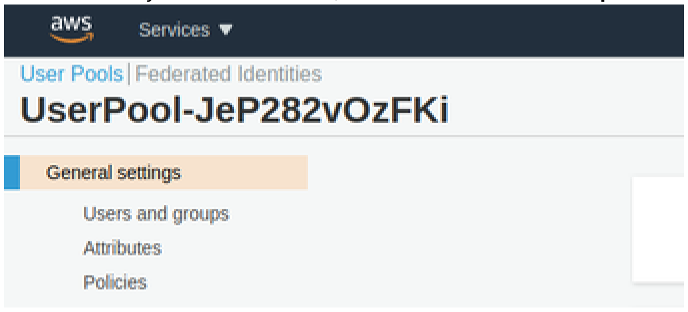
Head over to Cognito in the AWS console and click **Manager User Pools**.



Amazon Cognito

Amazon Cognito offers user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools provide AWS credentials to grant your users access to other AWS services.

Select the newly created User Pool, then click **Users and Groups**.

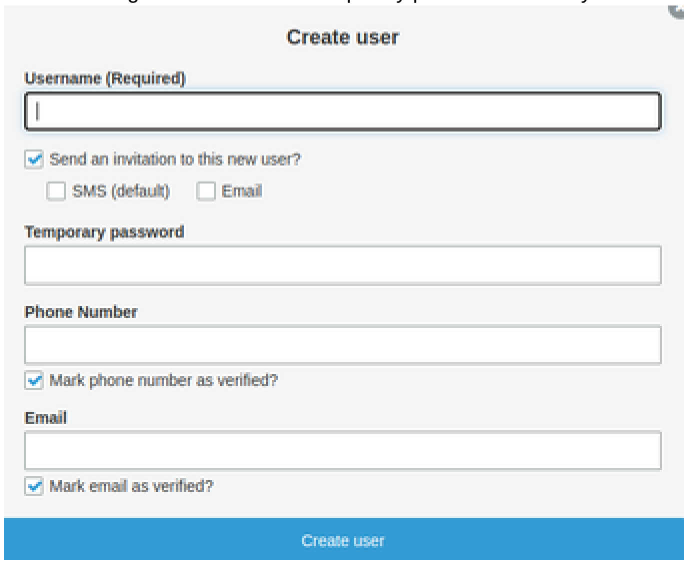


UserPool-JeP282vOzFKi

Click **Create User** and enter the user information.

- Username is required and should be the same as the user's email address.
- Temporary password is not required. One will be generated automatically if not supplied.
- Phone number is also not required.
- Email should be entered at a minimum. Uncheck the **Mark email as verified** checkbox.

The user will get an email with a temporary password that they will be forced to change upon first login.



Create user

Username (Required)

☑ Send an invitation to this new user?

☐ SMS (default) ☐ Email

Temporary password

Phone Number

☑ Mark phone number as verified?

Email

☑ Mark email as verified?

Once the user is created the user can login to the website. The site will be:

https://{subdomain}.myxillum.com

where the `subdomain` is the value entered when provisioning the DNS product.

> Although this user can now login, if the database data has not yet been ingested and transformed, nothing will be displayed in the charts and tables for the website after logging in.

Details

This product creates the following:

- A CloudFront distribution pointing at a specific version of the code Frontend code. The code is hosted in an S3 bucket in the root AWS account in a versioned folder.
- A DNS record for `subdomain.myxillum.com` pointing at the CloudFront distribution, where **subdomain** is the value entered previously.

Outputs

There are no outputs for this product.

Next

Next, provision the S3 Secure Upload Bucket.

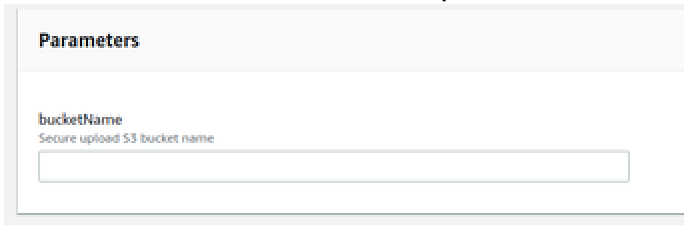
Next: S3 Secure Upload Bucket

Secure S3 Upload Bucket

Provision

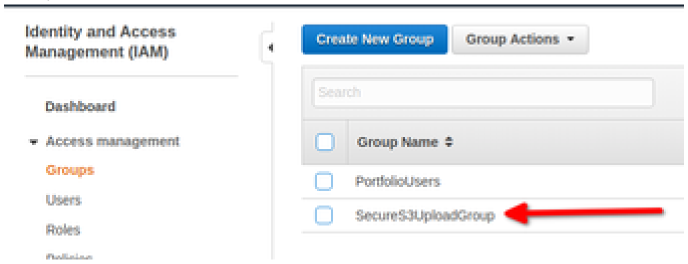
In the AWS Service Catalog Products list, click on **Xillum Secure S3 Upload Bucket**.

Enter a name for the bucket and click **Launch product**. A recommended name is something like: [customer]-secure-uploads


Parameters

bucketName
Secure upload S3 bucket name

Next, we need to create a user that can access this bucket. Over in IAM this product created an IAM group called **SecureS3UploadBucket**.



Create a new IAM user and add them to this group. This user will be able to login to the AWS console for this account and *only* add files to this S3 Bucket.

Details

The Secure S3 Upload Bucket product creates a bucket that will be used to allow the customer to upload HL7 files. This bucket has the following properties:

- Enryption at rest and in transit is enforced
- Versioning is turned on
- Public access is turned off; only explicitly authorized users can access this bucket.

Outputs

No outputs for this product.

Next

You have completed the account provisioning!

To Be Deleted

The following resources (created by RSI) can be safely deleted.

EC2

- xillum-dev-rdp
- xillum-dev-rdp-2
- xillum-dev-linux-1

CodePipeline

- ExpressApiPipeline

CodeCommit

- xillum-api

Cognito

All Cognito User Pools except `UserPool-QcxmCB1AsrWr` can be deleted.

- RyanTest
- RyanPagelTestPool01

The follow Cognito Identity Pools can be safely deleted:

- RyanPagelIdentityPool01

Database Overview

- RDS Configuration for New Client
- NextGen/Mirth Configuration

RDS Configuration for New Client

Create Xillum database.sql

NextGen/Mirth Configuration

HL7 Ingestion through Mirth Connect Administrator tool:

A Channel package named “Read HL7” was created using the Administrator tools through the Mirth Connect application.   This channel listens for HL7 files and ingestes them when they appear.   The HL7 file is parse through a destination channel using a HL7 Ver 7.2 transformer.   This transformer parses and assigns the data to the correct data column before the data is written to the MySQL database called “Xillum” and writes to the “InterfaceData” table.   This table feeds two PHP scripts that move that data as well as scheduling data from QGenda to populate the app front end.

This channel was tested using current data and the transformation occurs as expected. There was a QA check to see the finished data in the MySQL database table.   The PHP scripts could not be tested.

1. Download and install Mirth Connect Server Manager and Administrator software.
2. Use default setup configuration for Mirth Connect.
3. Download and install MySQL Workbench - latest version
4. Download and install PHP Release 7.4 from www.PHP.Net
5. Using Mirth Connect Administrator import channel using the Create HL7 Channel.XML file (attached)
6. Within the Read HL7 channel that is created edit the channel source for the correct connect string for your source file folder
7. Within the Read HL7 channel edit the Destination connector string, user and password to match your parameters
8. Edit both the UpdateHist and CleanData PHP scripts (attached) to match your connection string information
9. Edit the UpdateHist.PHP script to match your QGenda connection configuration.

This should give you a Mirth Connect channel that will listen for, download and parse your incoming HL7 files.

Create HL7 Channel.xml

updatehist - Comments.php

DataCleanup - Comments.php

PHP Script Recommendations

Within the attached files are comments and slightly modified code that we believe could improve performance on execution.

updatehist - Comments.php

Script modified with comments and a couple of functions to simplify some existing code. These changes have not been tested, but are examples of potential ways the script can be cleaned up.

There are notes within the script denoting some discrepancies in how some of the variables were being used. The use of a massive “multi_query()” at the end of the script is not ideal and would require extensive testing/research to determine how this could be optimized.

DataCleanup - Comments.php

Script modified with comments. There is some unused code that could be removed (denoted with comments). Another large “multi_query()” that needs more research/testing.

updatehist - Comments.php

DataCleanup - Comments.php