Running applications on YARN involves several key components and steps. Below, I'll outline a more detailed process, including a sample script to illustrate how you might submit and manage an application on a YARN cluster.

Overview of Running Applications on YARN

1. **Application Submission:**
   - The first step is to submit your application to YARN. This is typically done using command-line tools or through APIs provided by YARN.
2. **ResourceManager Allocation:**
   - The ResourceManager (RM) then takes over, allocating resources and scheduling the application's tasks across the cluster.
3. **ApplicationMaster (AM) Initialization:**
   - For each application, YARN starts an instance of the ApplicationMaster. The AM is responsible for managing the application's lifecycle and resources.
4. **Task Execution by NodeManagers:**
   - The ApplicationMaster communicates with the NodeManagers (NMs) on various nodes to execute the tasks of the application.
5. **Monitoring and Management:**
   - Throughout the execution, the status of the application can be monitored via the YARN ResourceManager UI or through command-line tools.

Example: Running a Custom Application on YARN

Prerequisites:

- A YARN-enabled Hadoop cluster.
- A user-defined application (could be MapReduce, Spark, etc.).
- Necessary permissions to submit jobs to the cluster.

Steps and Script Example:

1. **Prepare Your Application:**
   - Ensure your application (like a Java program or a Spark job) is ready and packaged appropriately (like a JAR file for Java applications).
2. **Write a YARN Client Script:**
   - This script interacts with the YARN API to submit the application. For a Java-based application, you would use the YARN client libraries.
3. **ResourceManager Submission:**
   - The script sends a request to the ResourceManager to run the application.
   - It specifies resource requirements, such as memory and CPU.
4. **ApplicationMaster Execution:**

- YARN starts an instance of the ApplicationMaster for your application.
- The AM negotiates resources and manages task execution.

5. **NodeManager Task Execution:**
   - The NodeManagers execute tasks as directed by the ApplicationMaster.
6. **Monitoring and Logging:**
   - Use YARN's ResourceManager UI or command-line tools to monitor the application's progress and troubleshoot if needed.

Example Client Script (Pseudo-code):

```
// Pseudo-code for a YARN client application
YarnClient yarnClient = YarnClient.createYarnClient();
yarnClient.init(conf);

// Start the client
yarnClient.start();

// Create application creation request
YarnClientApplication app = yarnClient.createApplication();
GetNewApplicationResponse appResponse = app.getNewApplicationResponse();

// Set up the container launch context for the application master
ContainerLaunchContext amContainer = ...;
amContainer.setCommands(...);

// Set up resource type requirements for ApplicationMaster
Resource capability = Resource.newInstance(1024, 1);
capability.setMemorySize(1024);
capability.setVirtualCores(1);

// Create the application submission context
ApplicationSubmissionContext appContext = app.getApplicationSubmissionContext();
appContext.setApplicationName("MyApplication");
appContext.setAMContainerSpec(amContainer);
appContext.setResource(capability);
appContext.setQueue("default");

// Submit the application
ApplicationId appId = appContext.getApplicationId();
System.out.println("Submitting application " + appId);
yarnClient.submitApplication(appContext);

// Monitor the application
while (true) {
```

```
    // Check application status
    ApplicationReport report = yarnClient.getApplicationReport(appId);
    YarnApplicationState state = report.getYarnApplicationState();
    if (state == YarnApplicationState.FINISHED ||
        state == YarnApplicationState.KILLED ||
        state == YarnApplicationState.FAILED) {
        break;
    }

    // Other monitoring and logging logic
    ...
}

yarnClient.stop();
```