1. Code

```
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;

import java.util.*;

public class CustomSchedulingSimulation {
    public static void main(String[] args) {
        // Initialize the CloudSim simulation environment
        int numUsers = 1;
        Calendar calendar = Calendar.getInstance();
        CloudSim.init(numUsers, calendar, false);

        // Create a datacenter
        Datacenter datacenter = createDatacenter("Datacenter_0");

        // Create a broker
        DatacenterBroker broker = createBroker();

        // Set the custom VM allocation policy
        VmAllocationPolicy policy = new CustomSchedulingPolicy(datacenter.getHostList());
        broker.setDatacenter(datacenter);
        broker.setVmAllocationPolicy(policy);

        // Create and submit cloudlets to the broker
        int numVMs = 5;
        int numCloudlets = 10;
        createVMsAndCloudlets(broker, numVMs, numCloudlets);

        // Start the simulation
        CloudSim.startSimulation();

        // Process the results and generate output
        List<Cloudlet> finishedCloudlets = broker.getCloudletReceivedList();

        // Stop the simulation
        CloudSim.stopSimulation();

        // Display the results
        printResults(finishedCloudlets);
    }

    private static Datacenter createDatacenter(String name) {
        List<Host> hostList = new ArrayList<>();

        // Create hosts with required characteristics
        for (int i = 0; i < 3; i++) {
            int mips = 1000; // Example MIPS value
            int ram = 2048; // Example RAM value
            long storage = 1000000; // Example storage value
            int bw = 10000; // Example bandwidth value

            hostList.add(new Host(i, new RamProvisionerSimple(ram), new
BwProvisionerSimple(bw),
                    storage, new ArrayList<>(), new VmSchedulerSpaceShared(new
ArrayList<>())));
        }

        // Create Datacenter Characteristics and return a Datacenter object
        String arch = "x86";
        String os = "Linux";
        String vmm = "Xen";
        double timeZone = 10.0;
        double cost = 3.0;
        double costPerMem = 0.05;
        double costPerStorage = 0.001;
        double costPerBw = 0.0;

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(arch, os,
vmm,
                hostList, timeZone, cost, costPerMem, costPerStorage, costPerBw);

        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList),
                    new ArrayList<>(), 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return datacenter;
    }

    private static DatacenterBroker createBroker() {
        DatacenterBroker broker = null;
        try {
            broker = new DatacenterBroker("Broker");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return broker;
    }

    private static void createVMsAndCloudlets(DatacenterBroker broker, int numVMs, int
numCloudlets) {
        List<Vm> vmList = new ArrayList<>();
        List<Cloudlet> cloudletList = new ArrayList<>();

        // Create VMs with required characteristics
        for (int i = 0; i < numVMs; i++) {
            int mips = 1000; // Example MIPS value
            int ram = 512; // Example RAM value
            long size = 10000; // Example storage value
            int bw = 1000; // Example bandwidth value
            int pesNumber = 1;

            Vm vm = new Vm(i, broker.getId(), mips, pesNumber, ram, bw, size, "Xen",
                    new CloudletSchedulerTimeShared());
            vmList.add(vm);
        }

        // Create cloudlets with required characteristics
        for (int i = 0; i < numCloudlets; i++) {
            long length = 10000; // Example cloudlet length
            int pesNumber = 1;
            long fileSize = 300;
            long outputSize = 300;
            UtilizationModel utilizationModel = new UtilizationModelFull();
            Cloudlet cloudlet = new Cloudlet(i, length, pesNumber, fileSize, outputSize,
                    utilizationModel, utilizationModel, utilizationModel);
            cloudlet.setUserId(broker.getId());
            cloudletList.add(cloudlet);
            broker.bindCloudletToVm(cloudlet.getCloudletId(), vmList.get(i % numVMs).getId());
// Assign VMs to cloudlets
        }

        broker.submitVmList(vmList);
        broker.submitCloudletList(cloudletList);
    }

    private static void printResults(List<Cloudlet> cloudlets) {
        // Process and print the simulation results
        // Display performance metrics like makespan, resource utilization, response time, etc.
        for (Cloudlet cloudlet : cloudlets) {
            System.out.println("Cloudlet ID: " + cloudlet.getCloudletId() +
                    ", VM ID: " + cloudlet.getVmId() +
                    ", Status: " + cloudlet.getStatus() +
                    ", Start Time: " + cloudlet.getExecStartTime() +
                    ", Finish Time: " + cloudlet.getFinishTime());
        }
    }
}
```

2. Code

```
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;

import java.util.*;

public class ResourceManagementSimulation {
    public static void main(String[] args) {
        int numUsers = 1;
        Calendar calendar = Calendar.getInstance();
        CloudSim.init(numUsers, calendar, false);

        Datacenter datacenter = createDatacenter("Datacenter");
        DatacenterBroker broker = createBroker();

        int numVMs = 20;
        List<Vm> vmList = createVMs(numVMs);

        int numCloudlets = 50;
        List<Cloudlet> cloudletList = createCloudlets(numCloudlets);

        broker.submitVmList(vmList);
        broker.submitCloudletList(cloudletList);

        CloudSim.startSimulation();
        CloudSim.stopSimulation();

        List<Cloudlet> finishedCloudlets = broker.getCloudletReceivedList();
        printResults(datacenter, finishedCloudlets);
    }

    private static Datacenter createDatacenter(String name) {
        // Create and configure the datacenter
        // Use classes like DatacenterCharacteristics, Host, VmAllocationPolicy, etc.
        // Return the created Datacenter object
        return null; // Replace null with your implementation
    }

    private static DatacenterBroker createBroker() {
        // Create and configure the broker
        // Use the DatacenterBroker class
        // Return the created DatacenterBroker object
        return null; // Replace null with your implementation
    }

    private static List<Vm> createVMs(int numVMs) {
        // Create and configure the virtual machines (VMs)
        // Set VM properties like MIPS, RAM, storage, and bandwidth
        // Return the list of created VMs
        return null; // Replace null with your implementation
    }

    private static List<Cloudlet> createCloudlets(int numCloudlets) {
        // Create and configure the cloudlets
        // Return the list of created cloudlets
        // Set cloudlet properties like length, utilization model, and data transfer size
        return null; // Replace null with your implementation
    }

    private static void printResults(Datacenter datacenter, List<Cloudlet> cloudlets) {
        // Process and print the results
        // Analyze the finished cloudlets and generate desired output
        System.out.println("Total simulation time: " + CloudSim.clock() + " seconds");
        System.out.println("Simulation Results");
        System.out.println("Datacenter Information:");
        System.out.println("Number of hosts: " + datacenter.getHostList().size());
        System.out.println("Number of virtual machines: " + datacenter.getVmList().size());
        System.out.println("- Number of cloudlets: " + cloudlets.size());
        System.out.println("Resource Utilization:");
        // Calculate and print average resource utilization
        System.out.println("- Average CPU utilization: " +
calculateAverageCPUUtilization(datacenter) + "%");
        System.out.println("- Average RAM utilization: " +
calculateAverageRAMUtilization(datacenter) + "%");
        System.out.println("- Average bandwidth utilization: " +
calculateAverageBandwidthUtilization(datacenter) + "%");
        System.out.println("Performance Metrics");
        // Calculate and print performance metrics
        System.out.println("- Makespan: " + calculateMakespan(cloudlets) + " seconds");
        System.out.println("- Total energy consumption: " +
calculateTotalEnergyConsumption(datacenter) + " joules");
        System.out.println("- Average response time: " +
calculateAverageResponseTime(cloudlets) + " seconds");
        System.out.println("- Throughput: " + calculateThroughput(cloudlets) + "
cloudlets/second");
    }
    private static double calculateAverageCPUUtilization(Datacenter datacenter) {
        // Calculate average CPU utilization
        double totalCPUUtilization = 0;
        for (Host host : datacenter.getHostList()) {
            totalCPUUtilization += host.getUtilizationOfCpu(CloudSim.clock());
        }
        return totalCPUUtilization / datacenter.getHostList().size();
    }
```

```
        }
    private static double calculateAverageRAMUtilization(Datacenter datacenter) {
        // Calculate average RAM utilization
        double totalRAMUtilization = 0;
        for (Host host : datacenter.getHostList()) {
            totalRAMUtilization += host.getUtilizationOfRam(CloudSim.clock());
        }
        return totalRAMUtilization / datacenter.getHostList().size();
    }
    private static double calculateAverageBandwidthUtilization(Datacenter datacenter) {
        // Calculate average bandwidth utilization
        double totalBandwidthUtilization = 0;
        for (Host host : datacenter.getHostList()) {
            totalBandwidthUtilization += host.getUtilizationOfBw(CloudSim.clock());
        }
        return totalBandwidthUtilization / datacenter.getHostList().size();
    }
    private static double calculateMakespan(List<Cloudlet> cloudlets) {
        // Calculate makespan
        double maxFinishTime = 0;
        for (Cloudlet cloudlet : cloudlets) {
            maxFinishTime = Math.max(maxFinishTime, cloudlet.getFinishTime());
        }
        return maxFinishTime;
    }
    private static double calculateTotalEnergyConsumption(Datacenter datacenter) {
        // Calculate total energy consumption
        double totalEnergyConsumption = 0;
        for (Host host : datacenter.getHostList()) {
            totalEnergyConsumption += host.getPowerModel().getEnergy(CloudSim.clock());
        }
        return totalEnergyConsumption;
    }
    private static double calculateAverageResponseTime(List<Cloudlet> cloudlets) {
        // Calculate average response time
        double totalResponseTime = 0;
        for (Cloudlet cloudlet : cloudlets) {
            totalResponseTime += cloudlet.getFinishTime() - cloudlet.getSubmissionTime();
        }
        return totalResponseTime / cloudlets.size();
    }
    private static double calculateThroughput(List<Cloudlet> cloudlets) {
        // Calculate throughput
        return cloudlets.size() / CloudSim.clock();
    }
}
```

3. Code

```
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;

import java.util.*;

public class LogForensicsSimulation {
    public static void main(String[] args) {
        int numUsers = 1;
        Calendar calendar = Calendar.getInstance();
        CloudSim.init(numUsers, calendar, false);

        List<LogEntry> logData = generateLogData();
        List<LogEntry> suspiciousActivities = detectSuspiciousActivities(logData);
        List<LogEntry> anomalies = detectAnomalies(logData);

        printSuspiciousActivities(suspiciousActivities);
        printAnomalies(anomalies);
    }

    private static List<LogEntry> generateLogData() {
        // Generate or retrieve log data for the simulation.
        // Simulate log entries with various attributes like timestamp, source IP, destination IP,
log message, etc.
        // Return the generated log data as a list of LogEntry objects
        return null; // Replace null with your implementation
    }

    private static List<LogEntry> detectSuspiciousActivities(List<LogEntry> logData) {
        // Implement log analysis algorithms to detect suspicious activities.
        // Use pattern matching, machine learning, statistical analysis, etc.
        // Return the list of detected suspicious activities as LogEntry objects
        return null; // Replace null with your implementation
    }

    private static List<LogEntry> detectAnomalies(List<LogEntry> logData) {
        // Implement log analysis algorithms to detect anomalies.
        // Use pattern matching, machine learning, statistical analysis, etc.
        // Return the list of detected anomalies as LogEntry objects
        return null; // Replace null with your implementation
    }

    private static void printSuspiciousActivities(List<LogEntry> suspiciousActivities) {
        // Print or process the list of detected suspicious activities.
        // Generate alerts, reports, or visualizations based on the detected activities
        System.out.println("Detected Suspicious Activities:");
        for (LogEntry entry : suspiciousActivities) {
            System.out.println(entry.toString());
        }
    }

    private static void printAnomalies(List<LogEntry> anomalies) {
        // Print or process the list of detected anomalies.
        // Generate alerts, reports, or visualizations based on the detected anomalies
        System.out.println("Detected Anomalies:");
        for (LogEntry entry : anomalies) {
            System.out.println(entry.toString());
        }
    }
}

class LogEntry {
    private String timestamp;
    private String sourceIP;
    private String destinationIP;
    private String logMessage;

    public LogEntry(String timestamp, String sourceIP, String destinationIP, String
logMessage) {
        this.timestamp = timestamp;
        this.sourceIP = sourceIP;
        this.destinationIP = destinationIP;
        this.logMessage = logMessage;
    }

    @Override
    public String toString() {
        return "Timestamp: " + timestamp + ", Source IP: " + sourceIP + ", Destination IP: " +
destinationIP +
                ", Log Message: " + logMessage;
    }
}
```

4. Code

```
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;

import java.util.*;

public class SecureFileSharingSimulation {
    public static void main(String[] args) {
        int numUsers = 1;
        Calendar calendar = Calendar.getInstance();
        CloudSim.init(numUsers, calendar, false);

        Datacenter datacenter = createDatacenter("Datacenter");
        List<User> users = createUsers(numUsers);
        associateUsersWithDatacenter(users, datacenter);

        List<FileRequest> fileRequests = generateFileRequests();
        for (FileRequest request : fileRequests) {
            User user = selectUser(users);
            byte[] fileData = generateFileData(request.getFileSize());
            uploadFile(user, request.getFileName(), fileData);
            byte[] downloadedFile = downloadFile(user, request.getFileName());
        }

        generateSimulationReport();
        generatePerformanceMetrics();
    }

    private static Datacenter createDatacenter(String name) {
        // Create and configure the datacenter
        // Use classes like DatacenterCharacteristics, Host, VmAllocationPolicy, etc.
        // Return the created Datacenter object
        return null; // Replace null with your implementation
    }

    private static List<User> createUsers(int numUsers) {
        // Create and configure user entities
        // Set up user properties, such as credentials, access privileges, etc.
        // Return the list of created User objects
        return null; // Replace null with your implementation
    }

    private static void associateUsersWithDatacenter(List<User> users, Datacenter
datacenter) {
        // Associate users with the datacenter
        // Create user entities and associate them with the datacenter
    }

    private static User selectUser(List<User> users) {
        // Implement user selection logic for file sharing activities
        // Choose a user from the list of available users based on a specific algorithm or criteria
        return null; // Replace null with your implementation
    }

    private static List<FileRequest> generateFileRequests() {
        // Implement the generation of file requests for simulation
        // Generate a list of file requests with properties like file name, size, etc.
        return null; // Replace null with your implementation
    }

    private static byte[] generateFileData(int fileSize) {
        // Generate random file data of the specified size for simulation
        return null; // Replace null with your implementation
    }

    private static void uploadFile(User user, String filename, byte[] fileData) {
        // Implement the secure file upload mechanism
        // Perform necessary security checks, encryption, and store the file in the cloud storage
    }

    private static byte[] downloadFile(User user, String filename) {
        // Implement the secure file download mechanism
        // Perform necessary security checks, decryption, and retrieve the file from the cloud
storage
        // Return the downloaded file data as a byte array
        return null; // Replace null with your implementation
    }

    private static void generateSimulationReport() {
        // Generate a report based on the simulation results
        // Include information on the file sharing activities, security aspects, and performance
metrics
    }

    private static void generatePerformanceMetrics() {
        // Generate performance metrics based on the simulation results
        // Calculate metrics like response time, throughput, security-related metrics, etc.
    }
}

class User {
    // Define User properties and methods
}

class FileRequest {
    // Define FileRequest properties and methods
}
```

5. Code

```
import pandas as pd

# Original dataset
data = pd.DataFrame({
   'Name': ['John Doe', 'Jane Smith', 'Michael Johnson'],
   'Email': ['johndoe@example.com', 'janesmith@example.com',
'michaeljohnson@example.com'],
   'Age': [25, 30, 35]
})

# Masking sensitive attributes
data['Name'] = 'XXXXXXXXXX'
data['Email'] = 'xxxxxxxxxxx'

# Output anonymized dataset
print(data)
```

K anymous code

```
import pandas as pd

# Original dataset
data = pd.DataFrame({
   'Name': ['John Doe', 'Jane Smith', 'Michael Johnson'],
   'Zip Code': ['12345', '67890', '54321'],
   'Age': [25, 30, 35]
})

# K-anonymization with generalization
data['Name'] = 'Anonymous'
data['Zip Code'] = 'XXXXX'

# Output anonymized dataset
print(data)
```

6. Code

```
import os
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import boto3

# Set AWS S3 credentials and bucket name
AWS_ACCESS_KEY_ID = 'the_access_key'
AWS_SECRET_ACCESS_KEY = 'the_secret_access_key'
BUCKET_NAME = 'the_bucket_name'

# Set encryption key (must be 16, 24, or 32 bytes long)
encryption_key = b'ThisIsASecretKey'
```

```
# Function to encrypt image
def encrypt_image(input_file):
    # Read the image file
    with open(input_file, 'rb') as file:
        image_data = file.read()

    # Generate a random initialization vector (IV)
    iv = os.urandom(16)

    # Create an AES cipher object
    cipher = AES.new(encryption_key, AES.MODE_CBC, iv)

    # Pad the image data
    padded_data = pad(image_data, AES.block_size)

    # Encrypt the padded data
    encrypted_data = cipher.encrypt(padded_data)

    # Return encrypted data and IV
    return encrypted_data, iv

# Function to upload encrypted image to S3
def upload_encrypted_image(encrypted_data, iv, filename):
    # Create an S3 client
    s3 = boto3.client('s3',
                aws_access_key_id=AWS_ACCESS_KEY_ID,
                aws_secret_access_key=AWS_SECRET_ACCESS_KEY)

    # Upload encrypted data as an S3 object
    s3.put_object(Body=encrypted_data, Bucket=BUCKET_NAME, Key=filename)

    # Upload IV as a separate S3 object
    iv_filename = f"{filename}.iv"
    s3.put_object(Body=iv, Bucket=BUCKET_NAME, Key=iv_filename)

# Set the path to the image file
input_file = "original_image.jpg"
# Encrypt the image
encrypted_data, iv = encrypt_image(input_file)
# Set the filename for the encrypted image
filename = "encrypted_image.jpg"
# Upload the encrypted image to S3
upload_encrypted_image(encrypted_data, iv, filename)
print("Image encrypted successfully.")
print("Image uploaded to S3 successfully.")
```

7. Code

```
from google.cloud import vision

def obfuscate_image(image_path):
    # Authenticate with Google Cloud Vision API client
    client = vision.ImageAnnotatorClient()

    # Read the image file
    with open(image_path, 'rb') as image_file:
        content = image_file.read()

    # Create a Vision API image object
    image = vision.Image(content=content)

    # Apply blurring to obfuscate the image
    response = client.safe_search_detection(image=image)
    blurred_image = response.full_text_annotation

    # Save the obfuscated image
    output_path = 'obfuscated_image.jpg'
    with open(output_path, 'wb') as output_image:
        output_image.write(content)

    return output_path

# Set the path to the image file
image_path = 'original_image.jpg'

# Obfuscate the image
obfuscated_image_path = obfuscate_image(image_path)

# Print the path to the obfuscated image
print("Obfuscated image path:", obfuscated_image_path)
```

8. Code

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.secrets import SecretClient

# Set up Azure credentials and client
credential = DefaultAzureCredential()

# Define RBAC roles and associated permissions
roles = {
    'end_user': ['read'],
    'admin': ['read', 'write', 'delete'],
    'developer': ['read', 'write']
}

# Define user roles
user_roles = {
    'user1@example.com': 'admin',
    'user2@example.com': 'developer',
    'user3@example.com': 'end_user'
}

# Get the logged-in user's email (replace this with the authentication logic)
logged_in_user_email = 'user1@example.com'

# Check access based on user's role
def check_access(permission):
    if logged_in_user_email in user_roles:
        user_role = user_roles[logged_in_user_email]
        if permission in roles[user_role]:
            return True
    return False

# Example usage: checking if user can write
can_write = check_access('write')
print('User can write:', can_write)
```

9. Code

```
class AttributeAuthority:
    def get_attribute(self, user_id, attribute_name):
        # Implement logic to retrieve attribute value for the provided user_id and attribute_name
        # Example: Get the user's role from a database
        if attribute_name == 'role':
            return self.get_user_role_from_database(user_id)
        # Example: Get the user's department from an external service
        elif attribute_name == 'department':
            return self.get_user_department_from_external_service(user_id)
        # Add more conditions for other attributes as needed
        else:
            return None
    def check_access(self, user_id, resource_id, action):
        # Instantiate an instance of the attribute authority
        attribute_authority = AttributeAuthority()
        # Define access control policies
        access_control_policies = [
            {'resource': 'sales_data', 'action': 'read', 'role': 'Admin', 'department': None},
            {'resource': 'sales_data', 'action': 'write', 'role': 'Manager', 'department': None},
            {'resource': 'sales_data', 'action': 'read', 'role': None, 'department': 'Sales'},
            # Add more access control policies as needed
        ]
        # Get attribute values for the user
        user_role = attribute_authority.get_attribute(user_id, 'role')
        user_department = attribute_authority.get_attribute(user_id, 'department')

        # Check if the user has access based on the attributes
        for policy in access_control_policies:
            if policy['resource'] == resource_id and policy['action'] == action:
                if (policy.get('role') is None or policy['role'] == user_role) and \
                   (policy.get('department') is None or policy['department'] == user_department):
                    return True
        return False
# Example usage: checking if user with ID 'user1' can read the 'sales_data' resource
attribute_authority = AttributeAuthority()
can_read = attribute_authority.check_access('user1', 'sales_data', 'read')
print("User can read:", can_read)
```

10. Code

```
import boto3
def generate_incident(event, context):
    # Extract relevant information from the log event
    log_group = event['detail']['logGroup']
    log_stream = event['detail']['logStream']
    log_message = event['detail']['message']
    # Perform further processing or anomaly detection based on log data
    # Generate an incident in an incident management system
    incident_title = 'Anomaly Detected in Log Stream: {}'.format(log_stream)
    incident_description = 'Anomaly detected in log group: {}\nLog message:
{}'.format(log_group, log_message)
   # Send the incident details to an incident management system
    incident_management_service = boto3.client('incident-manager')
    incident_management_service.create_incident(
        title=incident_title,
        impact=1,  # Define the impact level of the incident
        urgency=1,  # Define the urgency level of the incident
        severity=1,  # Define the severity level of the incident
        description=incident_description
    )

# Example usage: This Lambda function is intended to be triggered by log events in
CloudWatch.
# The output will depend on the incident management system used and its integration with
the Lambda function.
```