National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Rahul Rajendra Shirsat
Student ID: 23296356

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Rahul Rajendra Shirsat
Student ID:	23296356
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	10th August 2025
Project Title:	Configuration Manual
Word Count:	941
Page Count:	6

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Rahul Rajendra Shirsat
Date:	10th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1 Introduction

The purpose of the following instructions is to walk a user step-by-step, through the configuration and deployment of the "Decentralized Cloud Computing: Tokenized Resource Allocation" system. These instructions will enable a technical user to recreate the entire development environment, deploy all elements of the system and use the core features of it, as demonstrated throughout the research project.

The first part of the setup process consists of four distinct phases that follow the project's own development lifecycle:

1. **Blockchain Environment Setup:** Configuring the local blockchain network and deploying the `CloudToken` smart contract.
2. **AWS Infrastructure Configuration:** Provisioning required cloud resources and security permissions through the AWS Command Line Interface (CLI).
3. **Orchestration Layer Setup:** Installing dependencies and configuring the Python scripts that mediate between the blockchain and cloud layers.
4. **System Execution and Testing:** Setting up the core allocation logic, monitoring dashboard, and automated test scenarios.

Following these instructions will produce a fully working proof-of-concept demonstrating token-gated, scalable allocation of cloud compute resources.

2 Prerequisites

Before beginning the setup process, ensure the following software is installed and accounts are accessible.

2.1 Software Requirements

- **Node.js and npm:** Required for the Truffle Suite. Installation guides are available on the official Node.js website.
- **Truffle Suite:** A development framework for Ethereum. Install it globally using npm:

```
npm install -g truffle

```

- **Ganache UI:** A personal blockchain for Ethereum development. Download and install the application from the Truffle Suite website.
- **Python 3:** Version 3.7 or higher is required for the orchestration scripts.
- **AWS CLI:** The Amazon Web Services Command Line Interface. Follow the official AWS documentation to install and configure it with your credentials.

2.2 Account Requirements

- **AWS Account:** An active AWS account with administrative privileges is necessary to create IAM roles, users, and provision the required resources (EC2, S3, etc.).

3 Phase 1: Blockchain Environment Setup

This phase focuses on deploying the `CloudToken` smart contract to a local Ganache blockchain instance.

3.1 Project Initialization

1. Create a new directory for the blockchain components named `cloud-token`.
2. Navigate into this directory and initialize a new Truffle project.

```
mkdir cloud-token
cd cloud-token
truffle init
```

This will create the necessary directory structure for contracts, migrations, and tests.

3.2 Smart Contract Deployment

1. Launch the Ganache UI application. A new workspace will be created, running a local blockchain on `http://127.0.0.1:7545`.
2. Inside the `contracts/` directory, create a file named `CloudToken.sol`. Populate it with the ERC-20 smart contract code.
3. Inside the `migrations/` directory, create a file named `2_deploy_cloud_token.js`. This script instructs Truffle how to deploy the `CloudToken` contract.
4. Compile the smart contract to generate its ABI and bytecode.

```
truffle compile
```

5. Deploy the compiled contract to your local Ganache network.

```
truffle migrate --network development
```

3.3 Recording Contract Details

Upon successful migration, the Truffle console will display the transaction hash of the deployment and, most importantly, the new contract's on-chain address. Record the contract address (e.g., `0xC9b76694bBE9Aef88979cB394f4DB26b2DD415C1`). This address is required to configure the orchestration scripts in a later phase.

4 Phase 2: AWS Infrastructure Configuration

This phase details the setup of all necessary AWS resources using the AWS CLI. Ensure your CLI is configured for the `eu-north-1` region.

4.1 IAM User and Role Setup

1. Create a trust policy file named `trust-policy.json`. This policy allows the EC2 service to assume the role you are about to create.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "ec2.amazonaws.com" },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Create the IAM role named `CloudResourceManager` using the trust policy.

```
aws iam create-role ^
  --role-name CloudResourceManager ^
  --assume-role-policy-document file://trust-policy.json
```

3. Attach the required AWS-managed policies to the new role. These policies grant the necessary permissions to manage EC2, S3, and CloudWatch resources.

```
aws iam attach-role-policy ^
  --role-name CloudResourceManager ^
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess

aws iam attach-role-policy ^
  --role-name CloudResourceManager ^
  --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess

aws iam attach-role-policy ^
  --role-name CloudResourceManager ^
  --policy-arn arn:aws:iam::aws:policy/CloudWatchFullAccess
```

4.2 EC2 and S3 Preparation

1. In the AWS Management Console for the `eu-north-1` region, create a new EC2 key pair. Name it `ec2keypair` and download the `.pem` file. Store it securely.

2. Create an S3 bucket to serve as a conceptual resource pool. The bucket name must be globally unique.

```
aws s3 mb s3://tokenized-storage-pool-<Your-Account-ID> ^
  --region eu-north-1
```

Replace `<Your-Account-ID>` with your unique AWS Account ID.

4.3 Auto Scaling Configuration

1. Create an EC2 Launch Template that defines the configuration for instances launched by the Auto Scaling Group.

```
aws ec2 create-launch-template --launch-template-name
   TokenizedLaunchTemplate --version-description "Initial version" --
   launch-template-data "{\"ImageId\":\"ami-09278528675a8d54e\",\"
   InstanceType\":\"t3.micro\",\"KeyName\":\"ec2keypair\",\"
   TagSpecifications\":[{\"ResourceType\":\"instance\",\"Tags\":[{\"Key
   \":\"Type\",\"Value\":\"Compute\"},{\"Key\":\"AllocatedBy\",\"Value\
   ":\"TokenSystem\"}]}]}" --region eu-north-1
```

2. Create the Auto Scaling Group. You will need to provide a valid subnet ID from your default VPC in the `eu-north-1` region.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name
   TokenAutoScalingGroup --launch-template "LaunchTemplateName=
   TokenizedLaunchTemplate,Version=1" --min-size 1 --max-size 5 --
   desired-capacity 1 --vpc-zone-identifier <Your-Subnet-ID> --tags Key
   =Type,Value=Compute,PropagateAtLaunch=true --region eu-north-1
```

3. Create a scale-out policy for the group. This policy will add one instance when triggered.

```
aws autoscaling put-scaling-policy ^
  --auto-scaling-group-name TokenAutoScalingGroup ^
  --policy-name ScaleOutPolicy ^
  --scaling-adjustment 1 ^
  --adjustment-type ChangeInCapacity ^
  --region eu-north-1
```

4. Record the `PolicyARN` from the output of the previous command.

5. Create a CloudWatch alarm to trigger the scale-out policy based on high CPU utilization.

```
aws cloudwatch put-metric-alarm --alarm-name ScaleOutOnHighCPU --metric
   -name CPUUtilization --namespace AWS/EC2 --statistic Average --
   period 60 --threshold 60 --comparison-operator GreaterThanThreshold
   --dimensions Name=AutoScalingGroupName,Value=TokenAutoScalingGroup -
   -evaluation-periods 2 --alarm-actions <Your-Policy-ARN> --region eu-
   north-1
```

Replace `<Your-Policy-ARN>` with the ARN you recorded in the previous step.

5 Phase 3: Orchestration Layer Setup

This phase involves configuring the Python scripts that connect the blockchain and cloud layers.

5.1 Dependency Installation

Create a new directory outside of the `cloud-token` folder for the Python scripts. Install the required Python libraries using pip.

```
pip install web3
pip install boto3
pip install Flask
```

5.2 Script Configuration

1. Create the three Python files: `allocate.py`, `app.py`, and `testing_scenarios.py`.
2. In all three scripts, locate the variable for the contract address and update it with the address you recorded from the Truffle migration: `0xC9b76694bBE9Aef88979cB394f4DB26b2DD415C1`.
3. In the Ganache UI, identify two accounts to use as the "admin" and "user" wallets. Copy their public addresses and private keys.
4. Update the wallet address and private key variables in the scripts:
   - In `allocate.py`, update `default_wallet` and `default_key`.
   - In `app.py`, update `user_wallet`.
   - In `testing_scenarios.py`, update `admin_wallet`, `admin_private_key`, `user_wallet`, and `user_private_key`.
5. Ensure the path to the contract's JSON ABI file (`cloud-token/build/contracts/CloudToken.json`) is correct relative to the location of your Python scripts.

Note: Hardcoding private keys is insecure and suitable only for a local development environment. For a production system, use a secure key management solution like environment variables or a secrets manager.

6 Phase 4: System Execution and Testing

This final phase describes how to run the system's components.

6.1 Running the Core Allocation Logic

To manually trigger the allocation of a single EC2 instance using the default wallet configured in the script, execute the following command:

```
python allocate.py
```

The script will check the token balance and, if sufficient, transfer 10 CTK and launch an EC2 instance. The console will display the progress, including the transaction hash and the new instance ID.

6.2 Running the Monitoring Dashboard

To launch the web-based monitoring dashboard, execute:

```
python app.py
```

Open a web browser and navigate to `http://127.0.0.1:5000/`. The page will display the current CTK balance of the configured user wallet and a list of all active EC2 instances that have been allocated by the system.

6.3 Executing Automated Test Scenarios

To run the full suite of automated tests that simulate different user funding scenarios, execute:

```
python testing_scenarios.py
```

This script will perform a series of operations:

1. Test allocation with a sufficient balance.
2. Test allocation with an insufficient (zero) balance.
3. Test allocation after the balance has been replenished.

The console will display detailed output for each scenario, confirming that the system behaves correctly under each condition.

7 Conclusion

This guide covered the step-by-step method for setting up and operating the tokenized cloud resource allocation system. Following these phased instructions, a user can deploy the `CloudToken` smart contract, configure the necessary AWS infrastructure, and run the orchestration scripts that bring the project into operation. This set of activities leaves a fully replicated and functioning proof-of-concept environment that can then be used for further experimentation and development. Upon succeeding with test scenarios, we can, therefore, validate the proper configuration and integration of all components.