PostgreSQL Security Best Practices

Securing Amazon Aurora PostgreSQL and Amazon RDS for PostgreSQL resources on AWS

First published August 28, 2024

Last updated August 28, 2024

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers, or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved.

Contents

Abstract and introduction 5
- Abstract 5
- Introduction 5

Shared responsibility model 6
- Security in the cloud alongside the security of the cloud 6

Infrastructure 6
- AWS Nitro–based instances 6
- Internodal encryption 7

IAM permissions and policies 7

Networking 9
- Network isolation 9
- VPC flow logs 9
- VPC endpoints for RDS API access 10

Encryption 10
- AWS KMS 10
- Encryption at rest 10
- Encryption in transit 11

Authentication and authorization 11
- Secrets Manager and password rotation 11
- Kerberos authentication 11
- IAM authentication 12

Auditing and monitoring 12
- CloudTrail integration 12
- Event notifications 12
- CloudWatch metrics and alerting 12
- Publish database logs to CloudWatch 13
- Database Activity Streams 13

aws

Intrusion detection and prevention 13
Configuration 14
Master user 14
Parameter groups 14
Patch management 14
Engine-specific security features 15
Roles and permissions 15
Extensions 16
AWS JDBC Driver for PostgreSQL 17
Encryption 17
Custom DNS resolution for outbound connections 18
Logging and auditing 18
Contributors 19
Further reading 19
Document revisions 19

Abstract and introduction

Abstract

Amazon Relational Database Service (Amazon RDS) provides a managed platform on which customers can run a variety of relational databases. This whitepaper outlines best practices for securing Amazon Aurora PostgreSQL-Compatible Edition and Amazon RDS for PostgreSQL resources from improper access, data leaks, deletion, natural disasters, and other calamities. The target audience for this whitepaper includes database administrators, enterprise architects, systems administrators, and developers who would like to run their database workloads on Amazon RDS.

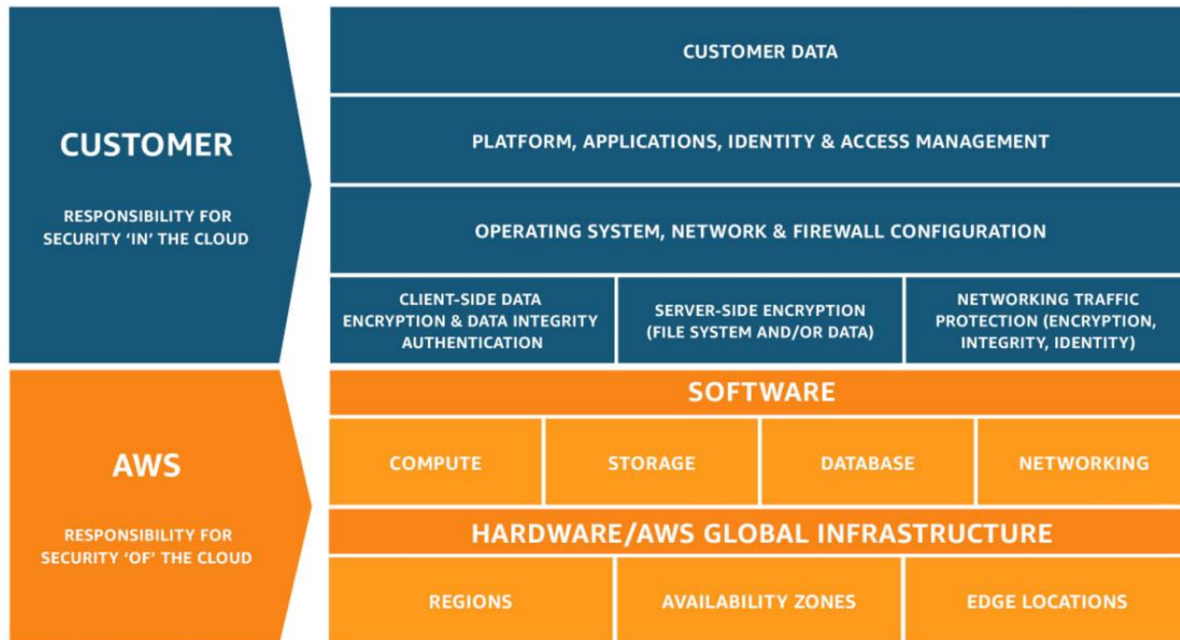
Introduction

Security is a critical facet of operating any information system. Securing data systems in the cloud is a shared responsibility between AWS and its customers. Customers benefit from the work that AWS does to secure the cloud platform on which customers operate, and they can choose to use additional services and features offered by AWS to further secure their data. However, customers must still secure the data systems they deploy in the cloud.

Security is complicated. To help you build and operate secure systems, the security pillar of the AWS Well-Architected Framework provides general guidance and best practices for designing, building, and operating secure systems in the cloud. This document expands upon that guidance for Amazon RDS PostgreSQL and Amazon Aurora PostgreSQL by diving specifically into the security features of the PostgreSQL database engine and the RDS and Amazon Aurora managed services. We also explore complementary and supporting services like Amazon Elastic Compute Cloud (Amazon EC2), Amazon Virtual Private Cloud (Amazon VPC), AWS Identity and Access Management (IAM), AWS CloudTrail, and Amazon CloudWatch.

Shared responsibility model

Security in the cloud alongside the security of the cloud



AWS implements a shared responsibility model with regard to resources in the cloud. In short, AWS is responsible for the security of the cloud, and customers are responsible for security in the cloud.

Security of the cloud – AWS provides secure global facilities and infrastructure that spans regions, availability zones and edge locations. Layered on top of this infrastructure are compute, storage, database, and networking resources that serve as the foundation for every service offered by AWS. From the data centers, all the way up to the software that manages these services, AWS provides a secure cloud on which our customers can build secure and compliant applications.

Security in the cloud – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations. AWS provides a set of features and services to help you secure your data. This paper helps you understand how to apply the shared responsibility model when using Amazon RDS.

Infrastructure

AWS Nitro–based instances

The AWS Nitro System breaks apart the traditional role of the hypervisor by providing dedicated hardware and software to control the CPU, storage, networking, bios, and other physical hardware. The net result is that virtually all of the actual server resources go toward running your workload rather than managing a hypervisor.

From a security perspective, the AWS Nitro system provides a specific security chip. This chip is locked down and does not allow administrative or human access, including from Amazon employees. Furthermore, this chip is constantly monitoring the security of the instance hardware and firmware.

For a full list of Amazon EC2 instances that are built on the Nitro system, see Instances built on the AWS Nitro System. The RDS instance types that use the AWS Nitro system include: T3, T4g, M7g, M6g, M5, R7g, R6i, R6g, R5, and Z1d. The Aurora instance types that use the Nitro system include: T3, T4g, R7g, R6g, R6i, R6gd, and R6id.

Internodal encryption

Communication between Aurora instances and Aurora storage is automatically encrypted and requires no additional configuration on the part of the end user.

IAM permissions and policies

IAM is the cornerstone of resource management on AWS. IAM defines the access permissions granted to entities to create, modify, and delete resources on AWS. In order to create an RDS instance, one must first have the appropriate IAM permissions to do so.

Permissions within IAM are defined using policies. A policy is a document outlining a certain set of operations (create, modify, delete) that can be applied to certain services or resources. Once a policy is defined, that policy can be attached to an IAM identity (user, group, or role). These identities can be assumed by people directly accessing AWS or by other resources on AWS. For example, it is common for an EC2 instance to assume a role that contains one or more policies. Let's say that one of those policies allows for writing files to an Amazon S3 bucket but does not allow for reading from the same bucket. In this case, application code running on that EC2 instance can generate log files and write them to the specified S3 bucket, but the application will not have access to read or delete those files. Similarly, let's say that a person has an AWS CloudFormation template that defines an Aurora cluster. In order to provision that template, the user will need to assume a role associated with a policy that grants permissions to create the cluster.

It is of critical importance to secure IAM identities and use restrictive policies. In order to secure IAM identities, it is best practice to only use the AWS account root user to create other users, and then lock away those root credentials. Likewise, when creating policies, it is best practice to only grant very narrow and specific permissions required to accomplish the task at hand. For example, if you wish to create a policy that allows a user to modify a single Aurora cluster, you can restrict access to the specific Amazon Resource Name (ARN) of that cluster. Or perhaps you would like to grant access to create new Aurora clusters but not to delete them. This can be controlled with a restrictive policy. This fine-grained access control is in addition to attaching the AmazonRDSDataFullAccess built-in policy to a given identity.

Beyond creating, modifying, and deleting database instances and clusters, IAM can also be used to authenticate application users to your Aurora PostgreSQL and RDS PostgreSQL databases. The general procedure can be summed up as follows:

1. Create or modify an Aurora PostgreSQL cluster or RDS PostgreSQL instance and set the ***Enable IAM DB Authentication*** parameter to "yes."
2. Create a local database user as follows:

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

3. Create an IAM policy that specifies connection rights to the user created in step 2. For example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "rds-db:connect"
      ],
      "Resource": [
"arn:aws:rds-db:region:account-id:dbuser:dbi-resource-id/database-user-name"
      ]
    }
  ]
}
```

region	AWS Region (for example, us-east-2)
account-id	AWS account ID (for example, 123456789012)
dbi-resource-id	Resource ID of the cluster (for example, cluster-OXA6X2XDA225H7PKKDB7FORNAY)
database-user-name	Local database user created in step 2

At this point, the database is configured to authenticate the user specified above using IAM authentication. The next step is to authenticate from your application. This authentication occurs using the SDK of your choice and calling the ***generate-db-auth-token*** method to get a temporary authentication token in lieu of a password. Subsequently, you would use the specified username that you created and the temporary authentication token as the password.

Note that using IAM authentication initially places additional load on your database instance. However, using connection pooling ensures that this load is only at application startup.

We recommend using IAM authentication when possible to remove the need for password management. There are, however, certain caveats to using IAM authentication.

Networking

Network isolation

The main construct available to users to control network access to RDS resources is Amazon VPC. RDS databases exist inside of a virtual private cloud (VPC). With a VPC, one can establish various subnets and define their connectivity to one another and the outside world. In nearly all circumstances, relational databases should be inaccessible, and they should not have access to network resources outside of the VPC. By adding your RDS databases to private subnets (with no direct route to an internet gateway), devices outside of your VPC do not have direct access to your RDS database. So long as there is no network address translation (NAT) device available to the subnet, your RDS databases do not have direct access to devices outside of your VPC.

VPCs also provide network access control lists (ACLs) to allow you to control network traffic at the subnet level. Traffic can be filtered based on protocol, port, and source.

A further component helping you control traffic is security groups. Security groups also allow you to control access at the network level, enabling traffic to be filtered based on protocol, port, and source. However, security groups also have the ability to filter another security group. For example, if you have a fleet of EC2 instances that need to communicate with your RDS instance, you can apply a security group named “ApplicationServers” to those EC2 instances and a second security group named “RdsResource” to your RDS database. To allow the EC2 instances to communicate with the RDS instance, you would add a rule to the RdsResource security group that allows ingress from the ApplicationServers security group. Adding a rule is more convenient than limiting access based solely on source IP ranges and provides more flexibility should underlying IP resources change.

Now you have deployed your RDS resources in a private subnet, inaccessible from external network devices, you need to administratively connect to those RDS resources. In this case, the most common approach is to use a bastion host. A bastion host is a compute resource that has access to both RDS resources and the outside world. In this scenario, although the bastion host is publicly accessible, the security group associated with this host should only allow access from known IP ranges. In turn, that security group will be granted access to the RDS resources that are not publicly accessible. From a user’s perspective, one can directly SSH to the bastion host and issue commands from the bastion host. Alternatively, a user can opt to create an SSH tunnel that will allow them to use local applications on their computer and tunnel the communication through the bastion host.

VPC flow logs

Once controls are put into place to manage network isolation, the next step is to audit network traffic. The primary tool to accomplish this is VPC flow logs. VPC flow logs identify network traffic flow. Flow logs

can be useful in diagnosing overly restrictive network ACLs or security groups, or they can help you uncover gaps in your network controls that are allowing traffic that should be prohibited.

VPC flow logs have no impact on network performance and can be published to Amazon CloudWatch Logs, Amazon Simple Storage Service (Amazon S3), or Amazon Data Firehose. Each of these destinations provides a different benefit. Amazon S3 is a cost-effective way to capture VPC flow logs as part of an audit trail that might be infrequently accessed. CloudWatch Logs provides a common logging location along with other log types, and it is straightforward to query from the AWS Management Console. Firehose is a great tool for ongoing, near real-time analysis of your VPC flow log records.

VPC endpoints for RDS API access

In some circumstances, you might run applications in your VPC that need the ability to provision, modify, or delete RDS resources. For example, you might run an EC2 instance that has assumed a role that gives it the ability to stop RDS instances in your development environment during off hours. Normally, the API call made to stop the RDS instance would need to travel on the public internet. If your EC2 instance is on a private subnet without an NAT gateway, it will not be able to connect to the RDS API to issue the command to stop the instance. By enabling AWS PrivateLink, your EC2 instance can now issue the stop command without being on a public subnet or using an NAT gateway. Traffic remains on the Amazon network, and neither your EC2 instance nor the relevant RDS instance is exposed to the public internet.

Encryption

AWS KMS

One of the most basic necessities of security when working with databases is to encrypt your data. To encrypt data requires encryption keys, and the management of those keys is a critical consideration when securing your databases. Fortunately, the AWS Key Management Service (AWS KMS) is specifically designed to create, manage, and rotate encryption keys, and it seamlessly integrates with the RDS platform. AWS KMS allows users to create symmetric and asymmetric keys for encryption and decryption purposes, as well as for HMAC authentication. AWS KMS is not specific to RDS and is integrated with a variety of other AWS services. This integration allows for a unified key management system across AWS. AWS KMS offers keys that are scoped to a single AWS Region to provide key isolation as well as keys that can be replicated between Regions for seamless cryptographic functionality around the globe.

Encryption at rest

Encryption at rest is straightforward to implement on the RDS platform. Whenever you create an Aurora cluster or RDS database instance, you need only check the box that indicates that your volume is to be encrypted, and then select the appropriate AWS KMS key to use for that encryption. From that point forward, the RDS platform will encrypt the entire database volume using the specified key. Snapshots and automated backups created from this volume will also be encrypted using the same key. You can choose a customer managed key that you have created or the AWS managed key. Using a customer managed key offers you greater control with regard to key rotation, key material origin, and Region. Additionally, if you

are planning to share an encrypted snapshot, it is possible to grant access to a customer managed key, but you cannot grant access to an AWS managed key.

Encryption in transit

Amazon Aurora PostgreSQL and RDS PostgreSQL support the use of SSL/TLS to encrypt communications between the database client software and the database host. Each support TLS versions 1.1, 1.2, and 1.3. You can force clients to connect using SSL/TLS encryption by setting the `rds.force_ssl` parameter to 1 in the database instance or cluster's parameter group. You can also force the use of specific TLS versions by setting values for `ssl_min_protocol_version` and `ssl_max_protocol_version` in the cluster's parameter group.

Amazon RDS Proxy also supports SSL/TLS to encrypt communications between the proxy and the database and between the database client and RDS Proxy. RDS Proxy supports TLS versions 1.0, 1.1, 1.2, and 1.3 (versions 1.0 and 1.1 are not recommended). You can force the use of SSL/TLS between RDS Proxy and the database with the Require Transport Layer Security setting when creating or updating the proxy. RDS Proxy uses SSL/TLS encryption between the client application and the proxy by default. Clients can override the default SSL mode using the `sslmode` connection parameter.

Clients can choose to verify the SSL certificate chain when connecting to Aurora PostgreSQL, RDS PostgreSQL, and RDS Proxy by setting the `sslmode` connection parameter.

For more information on encryption in transit, see Using SSL with a PostgreSQL DB instance for RDS, Securing Aurora PostgreSQL data with SSL/TLS for Aurora, and RDS Proxy security for RDS Proxy.

Authentication and authorization

Secrets Manager and password rotation

Although there are various ways to authenticate to a database instance, perhaps the most common is using a username and password. Often, these credentials are stored unencrypted in code or configuration files. This presents a significant security risk. If your application requires username and password authentication, you should consider storing those credentials in AWS Secrets Manager. Secrets Manager allows you to encrypt your credentials and then access them using an IAM role, as previously discussed in this whitepaper. Encryption ensures that usernames and passwords are never stored in plain text and are not embedded in application code. AWS provides tutorials on how to use Secrets Manager in your organization.

Kerberos authentication

Customers who use Microsoft Active Directory might wish to use Kerberos authentication with their RDS resources. Aurora PostgreSQL and RDS PostgreSQL both support Kerberos authentication. Kerberos authentication shifts the management of users and passwords to a centralized Microsoft Active Directory,

reducing user management work for database administrators. Kerberos authentication is a more secure approach to credential management than managing standalone usernames and passwords.

For more information, see Using Kerberos authentication with RDS for PostgreSQL and Using Kerberos authentication with Aurora PostgreSQL.

IAM authentication

Applications running on Amazon EC2, AWS Lambda, or other AWS compute services typically assume an IAM role that grants them access to other AWS services. Aurora PostgreSQL and RDS PostgreSQL support IAM authentication as an alternative to standard database authentication with passwords. IAM authentication does not use passwords, removing the risks associated with storing and using passwords. Instead, you use IAM credentials to generate a temporary token to use in place of a password when connecting.

For more information, see IAM database authentication for MariaDB, MySQL, and PostgreSQL for RDS and IAM database authentication for Aurora.

Auditing and monitoring

CloudTrail integration

CloudTrail records actions performed in an AWS account for operational and risk auditing, governance, and compliance. Actions can be initiated by a user, role, or AWS service while using the AWS console, command-line interface (CLI), and software development kits (SDKs). CloudTrail is automatically enabled and does not require any manual setup.

CloudTrail is an important part of RDS security, as it provides an audit mechanism for changes made to database resources. If, for example, an RDS instance is created and you need to know who created that instance, that information can be identified using CloudTrail. Additionally, CloudTrail provides the AWS CloudTrail Insights feature that helps you detect anomalies that CloudTrail discovers, alerting you to a potential security risk.

Event notifications

RDS event notification provides a mechanism to invoke an Amazon Simple Notification Service (Amazon SNS) topic when events related to RDS take place. For example, you might create an Amazon SNS topic that sends an email when invoked. You could then tie this SNS topic to an RDS event that is invoked when a database instance has an availability issue such as a shutdown or restart. Alternately, you can use a Lambda SNS endpoint, so that programmatic action takes place in response to the invoked event.

CloudWatch metrics and alerting

Amazon CloudWatch metrics are a critical component of managing any RDS database. Although most RDS metrics are performance related, when certain performance metrics such as CPU increase beyond what

is expected, it can be an indication of a bigger problem. CloudWatch metrics provide an alerting feature called Amazon CloudWatch alarms. These alarms can be configured like RDS events to invoke an SNS topic in response to CloudWatch metrics crossing specified thresholds.

By coupling the metrics captured by CloudWatch metrics with machine learning, Amazon DevOps Guru can automatically detect anomalies in workloads and, like CloudWatch alarms and RDS events, use SNS topics to respond to those anomalies.

Publish database logs to CloudWatch

RDS databases capture a variety of logs. Those logs vary by engine and are stored by default on storage local to the database instance. When working with a large number of database instances, it is not practical to connect to each instance to review its logs. Furthermore, processing the logs to find patterns or anomalies is also not a trivial undertaking. Fortunately, RDS offers the ability to publish database logs to Amazon CloudWatch Logs. By sending your logs to CloudWatch Logs, you now have a single, centralized repository to view logs. What's more, CloudWatch Logs offers the ability to query your logs through CloudWatch Logs Insights and detect anomalies in your logs using a log anomaly detector.

Database Activity Streams

Database Activity Streams in Amazon Aurora and RDS sends a near real-time feed of database audit events to Amazon Kinesis Data Streams for consumption by monitoring and compliance tools. Third-party tools like IBM's Security Guardium and Imperva's SecureSphere Database Audit and Protection can consume audit data from Kinesis Data Streams to monitor events occurring in the database. You can also build your own Kinesis data stream to process audit events. Activity streams are always encrypted. Database administrators don't have access to create, manage, or monitor activity streams unless they're explicitly granted those privileges through IAM. Once privileges are granted, you can monitor internal threats to your data systems.

For more information, see Monitoring RDS with Database Activity Streams and Monitoring Aurora with Database Activity Streams.

Intrusion detection and prevention

Amazon GuardDuty is an AWS service that can monitor CloudTrail event logs, CloudTrail management events, VPC flow logs, and DNS logs in an effort to detect communication with known bad actors and identify anomalous behavior. RDS is one of the many AWS services that GuardDuty supports. Aurora MySQL-Compatible and Aurora PostgreSQL are also currently supported. Aurora enables GuardDuty to access login activity to your database clusters, identify anomalous behavior, and report the details to you. This is a recommended feature to detect and prevent unauthorized access to your Aurora databases.

Master user

The master user is the closest login to root provided on an RDS instance. This user is created at the same time as an RDS instance and has the highest level of permissions available for the RDS instance. It is a recommended best practice to provide a strong password for this login, create other users with some subset of the master user's permissions, and then store the master user credentials in a secure location. This practice helps ensure the principle of least privilege and keeps your database secure.

Parameter groups

RDS parameter groups allow you to create a set of parameters that can be applied to one or many RDS instances of the same database engine type. For example, you might create a parameter group for PostgreSQL 16 that can then be applied to all PostgreSQL 16 instances in your account in the specified Region. You benefit from centralized management and the standardization of parameters. Inside of the parameter group, you can configure engine-specific security features. Those features are explored later in this whitepaper.

Patch management

Patch management is critical to database security. Over time, new patches are released for any given database engine that can contain performance or stability improvements, but often contain additional security fixes. In the case of RDS, patches are applied at the OS level and the database-engine level. In most cases, it is best practice to enable the automatic minor version upgrades feature. This feature ensures that your database instances are always kept up to date with the latest security patches. In order to minimize disruption to existing workloads, the minor version upgrade happens during your specified maintenance window.

Operating system updates come in two varieties. The first is optional updates. Get notified when an optional operating system update is available by using the RDS event functionality discussed earlier and specifically subscribing to RDS-EVENT-0230.

The second category of operating system updates is the mandatory category. Unlike optional updates, which can be skipped at your discretion, mandatory updates come with an apply date. If you do not apply the update by that date, it will be automatically applied for you during your specified maintenance window.

Roles and permissions

No superuser role

The PostgreSQL database requires a superuser role to boot the database system. The superuser has access to the underlying host operating system and is required to perform some administrative tasks. If you were to install PostgreSQL on an Amazon EC2 instance, you would have access to the superuser role. However, when you create a new Aurora PostgreSQL or RDS PostgreSQL cluster or instance, the RDS service boots the database and handles the administrator tasks that require superuser access. RDS does not give you access to the PostgreSQL superuser role, but provides you the more limited rds_superuser role instead. This more limited role keeps you from accidentally interfering with administrative work that RDS performs on your behalf, but it also means that you cannot compromise the security of the underlying host.

For more information, see Understanding PostgreSQL roles and permissions.

Restricted user password management

Roles in PostgreSQL can set and change their password and adjust their password lifetime constraint. Database administrators might wish to restrict that ability to normal users and delegate password management responsibilities to a controlled set of users. Aurora PostgreSQL and RDS PostgreSQL can restrict password management capabilities to roles with explicitly granted privileges.

To activate this privilege-control feature, use a custom parameter group for your database instance and set the value of the `rds.restrict_password_commands` parameter to 1, then reboot the instance. When activated, only roles that have been granted the `rds_password` role can set or change passwords, set or change password expiration, or rename roles.

We recommend granting `rds_password` privileges to only a few roles that you solely use for password management.

For more information, see Understanding PostgreSQL roles and permissions.

SCRAM password encryption

The Salted Challenge Response Authentication Mechanism (SCRAM) is an alternative to PostgreSQL's default message digest algorithm 5 (MD5) for password encryption. SCRAM is a cryptographic challenge-response mechanism that uses the scram-sha-256 algorithm for password authentication and encryption. SCRAM authentication improves the security of password-based user authentication by adding features that prevent rainbow-table attacks, man-in-the-middle attacks, and stored password attacks, while also adding support for multiple hashing algorithms and passwords that contain non-ASCII characters. RDS Proxy is compatible with SCRAM password authentication

AWS recommends using SCRAM rather than MD5 as the password encryption scheme for your RDS for PostgreSQL DB instance. The `rds.accepted_password_auth_method` configuration parameter can be set to force the use of SCRAM passwords. Older client libraries do not support SCRAM passwords, so it might be necessary to upgrade database drivers and clients.

For more information and instructions for setting up SCRAM encryption, see Using SCRAM for PostgreSQL password encryption.

Extensions

Allowed extensions list

Functionality can be added to a PostgreSQL database through extensions or modules developed by the open source community and installed on the database host. RDS PostgreSQL and Aurora PostgreSQL allow the use of extensions, with approved extensions already installed on the database instances by the RDS service.

Any database user with sufficient privileges can load one of the approved extensions using the CREATE EXTENSION command. Aurora PostgreSQL and RDS PostgreSQL provide the `rds.allowed_extensions` configuration parameter that allows database administrators to restrict which extensions can be loaded to a defined list. When `rds.allowed_extensions` is used, any attempt to load an extension that is not in the list will be denied.

For more information and configuration instructions, see Restricting installation of PostgreSQL extensions.

Delegated extension management

Aurora PostgreSQL supports delegated extension management, allowing users who do not have the `rds_superuser` role the ability to create and drop extensions. Only users with the `rds_extension` role are permitted to manage extensions, and the `rds.allowed_delegated_extensions` and `rds.allowed_extensions` configuration parameters restrict which extensions can be managed.

Delegated extension management provides two benefits. First, it reduces the workload of database administrators, allowing privileged users to self-serve in a controlled way. Second, it provides the ability to support different sets of extensions for different databases in the same database cluster.

For more information and setup instructions, see Using Aurora delegated extension support for PostgreSQL.

Trusted extensions

Installing extensions to a PostgreSQL database typically requires superuser privileges. PostgreSQL 13 introduced the concept of trusted extensions that can be installed without superuser privileges by users with CREATE EXTENSION privileges. Extensions must be marked as trusted by a superuser. Some built-in

extensions are marked trusted by default. In RDS PostgreSQL and Aurora PostgreSQL, trusted extensions reduce the need to use elevated `rds_superuser` privileges.

See PostgreSQL trusted extensions for a list of trusted extensions supported by RDS PostgreSQL and Aurora PostgreSQL.

TLE

Trusted Language Extensions (TLE) for PostgreSQL is an open source development kit for building PostgreSQL extensions. It allows you to build high-performance PostgreSQL extensions and safely run them on your RDS for PostgreSQL database instance. TLE provides a development environment for creating new extensions for any PostgreSQL database without the use of the host's filesystem, making it ideal for use in Aurora PostgreSQL and RDS PostgreSQL. TLE is designed to prevent access to unsafe resources for the extensions that you create using TLE. Its runtime environment limits the impact of any extension defect to a single database connection. TLE also gives database administrators fine-grained control over who can install extensions, and it provides a permissions model for running those extensions.

For more information, see Working with TLE for PostgreSQL.

AWS JDBC Driver for PostgreSQL

The AWS JDBC Driver for PostgreSQL extends the community pgJDBC driver for use with RDS for PostgreSQL, adding support for IAM database authentication and integration with Secrets Manager.

For more information, see Connecting to a DB instance running the PostgreSQL database engine.

Encryption

Required SSL/TLS encrypted connections

Aurora PostgreSQL and RDS PostgreSQL support encrypted secured sockets layer / transport layer security (SSL/TLS) client connections to the database. Database clients can choose whether or not they connect to the database over an encrypted connection. RDS offers database administrators the ability to force clients to connect over SSL/TLS using the `rds.force_ssl` parameter. When this parameter is set to 1, unencrypted attempts to connect to the database are rejected.

For more information, see Requiring an SSL connection to a PostgreSQL DB instance (RDS) and Requiring an SSL/TLS connection to an Aurora PostgreSQL DB cluster.

pgcrypto

Aurora PostgreSQL and RDS PostgreSQL support pgcrypto, an extension that provides cryptographic functions for PostgreSQL. The pgcrypto extension is trusted, so it can be loaded without requiring

rds_superuser privileges. With pgcrypto, you can encrypt and decrypt sensitive data like credit card numbers and Social Security numbers in your database.

Custom DNS resolution for outbound connections

RDS database instances can make outbound network connections. You can control how domain names are resolved by the RDS instance by setting up a custom domain name server in your VPC and instructing RDS to use it for name resolution. Setting the value of the `rds.custom_dns_resolution` parameter to 1 causes RDS to use any domain name servers referenced in the VPC's DHCP option set.

For more information and configuration instructions, see Using a custom DNS server for outbound network access.

Logging and auditing

pgAudit

Aurora PostgreSQL and RDS PostgreSQL support pgAudit, an extension that provides detailed audit records that can help you meet the requirements of regulators and auditors. The pgAudit extension builds on the functionality of the native PostgreSQL logging infrastructure by extending the log messages with more detail, allowing you to use the same method to view your audit log as you do to view database log messages. The pgAudit extension redacts sensitive data such as cleartext passwords from the logs.

For information about setting up and using pgAudit, see Using PostgreSQL extensions with RDS for PostgreSQL.

RDS internal user activity logging

RDS performs some database management tasks for you as part of its managed service offering. You can log the activities of the internal user (rdsadmin) performing those tasks as part of your auditing activities. To enable logging of internal admin activities, set the `rds.force_admin_logging_level` configuration parameter to one of the following values: `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, or `panic`. To disable internal user activity logging, set `rds.force_admin_logging_level` to `disabled`.

For more information, see Working with logging mechanisms supported by RDS for PostgreSQL.

Conclusion

This whitepaper outlines the most important security best practices for deploying and protecting Aurora PostgreSQL and RDS for PostgreSQL databases. By using RDS security features like encryption at rest, IAM authentication, and advanced auditing, organizations can safely run their critical relational workloads in the cloud.

Contributors

Contributors to this document include:

- Dan Blaner, Principal Database Specialist Solutions Architect, AWS
- Steve Abraham, Principal Database Specialist Solutions Architect, AWS

Further reading

For additional information, refer to:

- AWS Architecture Center
- PostgreSQL on RDS
- PostgreSQL on Aurora
- Security Pillar of the AWS Well-Architected Framework

Document revisions

Date	Description
August 28, 2024	First publication