aws

Developer Guide

AMB Access Polygon

AMB Access Polygon: Developer Guide

Table of Contents

Amazon Managed Blockchain (AMB) Access Polygon is in preview release and is subject to change.

What is Amazon Managed Blockchain (AMB) Access Polygon?

Amazon Managed Blockchain (AMB) Access Polygon is a fully managed service that helps you build resilient Web3 applications on the Polygon blockchain. AMB Access Polygon provides instant and serverless access to the Polygon blockchain.

Polygon is a scaling solution that uses the Ethereum Virtual Machine (EVM) as the foundation. The Polygon blockchain is known for high transaction throughput and low transaction fees. The Polygon blockchain uses a proof-of-stake consensus mechanism. Polygon is commonly used in building decentralized applications (dApps) related to NFTs, Web3 games, and tokenization use cases, among others.

This guide covers how to create and manage Polygon blockchain resources using Amazon Managed Blockchain (AMB) Access Polygon.

Resources for first-time AMB Access Polygon users

If this is your first time using AMB Access Polygon, we recommend that you begin by reading the following sections:

- Key concepts: Amazon Managed Blockchain (AMB) Access Polygon
- Getting started with Amazon Managed Blockchain (AMB) Access Polygon
- Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon

Key concepts: Amazon Managed Blockchain (AMB) Access Polygon

Note

This guide assumes that you're familiar with the concepts that are essential to Polygon. These concepts include staking, dApps, transactions, wallets, smart contracts, Polygon (POL, formerly MATIC), and others. Before using Amazon Managed Blockchain (AMB) Access Polygon, we recommend that you review the Polygon Development Documentation and the Polygon wiki.

Amazon Managed Blockchain (AMB) Access Polygon provides you with serverless access to the Polygon Mainnet and Polygon Mainnet networks, without requiring you to provision and manage any Polygon infrastructure, including nodes. Polygon nodes on a network collectively store a Polygon blockchain state, verify transactions, and participate in consensus to change a blockchain state. You can use this managed service to access the Polygon networks quickly and on demand, reducing your overall cost of ownership.

With AMB Access Polygon, you have access to JSON Remote Procedure (JSON-RPC) calls. You can invoke Polygon JSON-RPCs to communicate with the Polygon blockchain through nodes managed by Managed Blockchain. You can use the AMB Access Polygon service to develop and use decentralized applications (dApps) that interact with the Polygon blockchain. An integral part of dApps are *smart contracts*. You can create and deploy smart contracts into the Polygon blockchain using AMB Access Polygon. You can also check balances for your wallets, transaction details, estimate fees, and so on, by invoking JSON-RPCs against AMB Access Polygon endpoints that run in a decentralized way across all the nodes that are peers to the Polygon network. Any peer to the Polygon network can develop and deploy a smart contract.

Important

You are responsible for creating, maintaining, using, and managing your Polygon addresses. You are also responsible for the contents of your Polygon addresses. AWS is not responsible for any transactions deployed or called using Polygon nodes on Amazon Managed Blockchain.

Considerations and limitations for using Amazon Managed Blockchain (AMB) Access Polygon

When you use Amazon Managed Blockchain (AMB) Access Polygon, consider the following:

- **Supported Polygon networks**

  AMB Access Polygon supports the following public networks:

  - **Mainnet**—The public Polygon blockchain secured by proof-of-stake consensus, and on which the Polygon (POL) token is issued and transacted. Transactions on Mainnet have actual value (that is, they incur real costs) and are recorded on the public blockchain.

- **Networks no longer supported by Polygon**

  - As communicated by Polygon Labs, the Mumbai Testnet network will sunset in mid-April. In line with this news, AMB Access Polygon ended support of the Mumbai Testnet on April 15, 2024. We recommend using Amoy Testnet for your testing workload.
  - Private networks are not supported.
  - Furthermore, AMB Access Polygon does not include support for the *Polygon zkEVM* network.

- **Compatibility with popular third-party programming libraries**

  AMB Access Polygon is compatible with popular programming libraries, such as ethers.js, allowing developers to interact with the Polygon blockchain using familiar tools to integrate easily with their existing implementations or develop new applications quickly.

- **Supported Regions**

  This service is supported only in the US East (N. Virginia) Region.

- **Service endpoints**

  The following are the service endpoints for AMB Access Polygon. To connect with the service, you must use an endpoint that includes one of the supported Regions.

  - `mainnet.polygon.managedblockchain.us-east-1.amazonaws.com`

- **Staking not supported**

  AMB Access Polygon does not support Polygon (POL) validator nodes for proof-of-stake.

- **Signature Version 4 signing of Polygon JSON-RPC requests**

When making calls to the Polygon JSON-RPCs on Amazon Managed Blockchain, you can do so over an HTTPS connection authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account can make Polygon JSON-RPC calls. To do this, AWS credentials (an access key ID and a secret access key) must be provided with the call.

> **Important**
>
> - Do not embed client credentials in user-facing applications.
> - You cannot use IAM policies to restrict access to individual Polygon JSON-RPCs.

- **Support for Token Based Access**

  You can also use *Accessor* tokens to make JSON-RPC calls to the Polygon network endpoints as a convenient alternative to the Signature Version 4 (SigV4) signing process. You must provide a `BILLING_TOKEN` from one of the Accessor tokens you create and add as a parameter with your calls.

  > **Important**
  >
  > - If you prioritize security and auditability over convenience, use the SigV4 signing process instead.
  > - You can access the Polygon JSON-RPCs using Signature Version 4 (SigV4) and token-based access. However, if you choose to use both protocols, your request is rejected.
  > - You must never embed Accessor tokens in user-facing applications.

- **Only submissions of raw transactions are supported**

  Use the `eth_sendrawtransaction` JSON-RPC to submit transactions that update the Polygon blockchain state.

Setting up Amazon Managed Blockchain (AMB) Access Polygon

Before you use Amazon Managed Blockchain (AMB) Access Polygon for the first time, follow the steps in this section to create an AWS account. The following chapter discusses how to start using AMB Access Polygon.

Prerequisites for using AMB Access Polygon

Before you use AWS for the first time, you must have an AWS account.

Sign up for AWS

When you sign up for AWS, your AWS account is automatically signed up for all AWS services, including Amazon Managed Blockchain (AMB) Access Polygon. You're charged only for the services that you use.

If you have an AWS account already, go to the next step. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

Create an IAM user with appropriate permissions

To create and work with AMB Access Polygon, you must have an AWS Identity and Access Management (IAM) principal (user or group) with permissions that allow necessary Managed Blockchain actions.

When making calls to the Polygon JSON-RPCs on Amazon Managed Blockchain, you can do so over an HTTPS connection authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account can make Polygon JSON-RPC calls. To do this, AWS credentials (an access key ID and a secret access key) must be provided with the call.

You can also use *Accessor* tokens to make JSON-RPC calls to the Polygon network endpoints as a convenient alternative to the Signature Version 4 (SigV4) signing process. You must provide a `BILLING_TOKEN` from one of the Accessor tokens you create and add as a parameter with your calls. However, you still need IAM access to get permissions to create Accessor tokens using the AWS Management Console, AWS CLI, and SDK.

For information about how to create an IAM user, see Creating an IAM user in your AWS account. For more information about how to attach a permissions policy to a user, see Changing permissions for an IAM user. For an example of a permissions policy that you can use to give a user permission to work with AMB Access Polygon, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Polygon.

Install and configure the AWS Command Line Interface

If you have not already done so, install the latest AWS Command Line Interface (AWS CLI) to work with AWS resources from a terminal. For more information, see Installing or updating the latest version of the AWS CLI.

> **Note**
>
> For CLI access, you need an access key ID and a secret access key. Use temporary credentials instead of long-term access keys when possible. Temporary credentials include an access key ID, a secret access key, and a security token that indicates when the credentials expire. For more information, see Using temporary credentials with AWS resources in the *IAM User Guide*.

Getting started with Amazon Managed Blockchain (AMB) Access Polygon

Get started with Amazon Managed Blockchain (AMB) Access Polygon by using the information and procedures in this section.

Topics

Create an IAM policy to access the Polygon blockchain network

To access the public endpoint for the Polygon Mainnet to make JSON-RPC calls, you must have user credentials (`AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`) that have the appropriate IAM permissions for Amazon Managed Blockchain (AMB) Access Polygon. In a terminal with the AWS CLI installed, run the following command to create an IAM policy to access both Polygon endpoints:

```
cat <<EOT > ~/amb-polygon-access-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid" : "AMBPolygonAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:InvokeRpcPolygon*"
            ],
            "Resource": "*"
        }
    ]
}
EOT
```

```
aws iam create-policy --policy-name AmazonManagedBlockchainPolygonAccess --policy-
document file://$HOME/amb-polygon-access-policy.json
```

> **Note**
>
> The previous example gives you access to all available Polygon networks. To get access to a specific endpoint, use the following `Action` command:
>
> - `"managedblockchain:InvokeRpcPolygonMainnet"`

After you create the policy, attach that policy to your IAM user's role for it to take effect. In the AWS Management Console, navigate to the IAM service, and attach the policy `AmazonManagedBlockchainPolygonAccess` to the role assigned to your IAM user.

Make Polygon remote procedure call (RPC) requests on the AMB Access RPC editor using the AWS Management Console

You can edit, configure, and submit remote procedure calls (RPCs) on the AWS Management Console using AMB Access Polygon. With these RPCs, you can read data and write transactions on the Polygon network, including retrieving data and submitting transactions to the Polygon network.

Example

The following example shows how to get information about the *latest* block by using `eth_getBlockByNumber` RPC. Change the highlighted variables to your own inputs or choose one of the **RPC methods** listed and enter in the relevant inputs required.

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **RPC editor**.
3. In the **Request** section, choose *`POLYGON_MAINNET`* as the ***`Blockchain Network`***.
4. Choose *`eth_getBlockByNumber`* as the **RPC method**.
5. Enter **`latest`** as the ***`Block number`*** and choose *`False`* as the **Full transaction flag**.
6. Then, choose **Submit RPC**.

7. You get the results of the `latest` block in the **Response** section. You can then copy the full raw transactions for further analysis or to use in business logic for your applications.

For more information, see the RPCs supported by AMB Access Polygon

Make AMB Access Polygon JSON-RPC requests in `awscurl` by using the AWS CLI

Example

Sign requests with your IAM user credentials by using Signature Version 4 (SigV4) in order to make Polygon JSON-RPC requests to the AMB Access Polygon endpoints. The `awscurl` command line tool can help you sign requests to AWS services using SigV4. For more information, see the awscurl README.md.

Install `awscurl` by using the method appropriate to your operating system. On macOS, HomeBrew is the recommended application:

```
brew install awscurl
```

If you have already installed and configured the AWS CLI, your IAM user credentials and the default AWS Region are set in your environment and have access to `awscurl`. Using `awscurl`, submit a request to the Polygon *Mainnet* by invoking the `eth_getBlockByNumber` RPC. This call accepts a string parameter corresponding to the block number for which you want to retrieve information.

The following command retrieves the block data from the Polygon Mainnet by using the block number in the `params` array to select the specific block for which to retrieve the headers.

```
awscurl -X POST -d '{ "jsonrpc": "2.0", "id": "eth_getBlockByNumber-curltest",
 "method":"eth_getBlockByNumber", "params":["latest", false] }' --service
 managedblockchain https://mainnet.polygon.managedblockchain.us-east-1.amazonaws.com -k
```

> **Tip**
>
> You can also make this same request using `curl` and the AMB Access token based access feature using `Accessor` tokens. For more information, see Creating and managing Accessor tokens for token-based access to make AMB Access Polygon requests.

```
curl -X POST -d '{"jsonrpc":"2.0", "id": "eth_getBlockByNumber-curltest",
 "method":"eth_getBlockByNumber", "params":["latest", false] }'
 'https://mainnet.polygon.managedblockchain.us-east-1.amazonaws.com?
billingtoken=your-billing-token'
```

The response from either command returns information about the *latest* block. See the following example for illustrative purposes:

```
{"error":null,"id":"eth_getBlockByNumber-curltest","jsonrpc":"1.0",
      "result":{"baseFeePerGas":"0x873bf591e","difficulty":"0x18",
      "extraData":"0xd783010006836 26f7288676f312e32312e32856c696e757800000000000000000009a
\
      423a58511085d90eaf15201a612af21ccbf1e9f8350455adaba0d27eff0ecc4133e8cd255888304cc
\
      67176a33b451277c2c3c1a6a6482d2ec25ee1573e8ba000",
      "gasLimit":"0x1c9c380","gasUsed":"0x14ca04d",
      "hash":"0x1ee390533a3abc3c8e1306cc1690a1d28d913d27b437c74c761e1a49********;",
      "nonce":"0x0000000000000000","number":"0x2f0ec4d",

 "parentHash":"0x27d47bc2c47a6d329eb8aa62c1353f60e138fb0c596e3e8e9425de163afd6dec",

 "receiptsRoot":"0x394da96025e51cc69bbe3644bc4e1302942c2a6ca6bf0cf241a5724c74c063fd",

 "sha3Uncles":"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
      "size":"0xbd6b",
      "stateRoot":"0x7ca9363cfe9baf4d1c0dca3159461b2cca8604394e69b30af05d7d5c1beea6c3",
      "timestamp":"0x653ff542",
      "totalDifficulty":"0x33eb01dd","transactions":[...],

 "transactionsRoot":"0xda1602c66ffd746dd470e90a47488114a9d00f600ab598466ecc0f3340b24e0c",
      "uncles":[]}}
```

Make Polygon JSON-RPC requests in Node.js

You can invoke Polygon JSON-RPCs by submitting signed requests using HTTPS to access the Polygon *Mainnet* network using the native https module in Node.js, or you can use a third-party library such as AXIOS. The following *Node.js* examples show you how to make Polygon JSON-RPC

requests to the AMB Access Polygon endpoint using both Signature Version 4 (SigV4) and token-based access. The first example sends a transaction from one address to another and the following example requests transaction details and balance information from the blockchain.

Example

To run this example Node.js script, apply the following prerequisites:

1. You must have node version manager (nvm) and Node.js installed on your machine. You can find installation instructions for your OS here.
2. Use the `node --version` command and confirm that you are using *Node version 18* or higher. If required, you can use the `nvm install v18.12.0` command, followed by the `nvm use v18.12.0` command, to install *version 18*, the *LTS* version of Node.
3. The environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` must contain the credentials that are associated with your account. .

   Export these variables as strings on your client by using the following commands. Replace the values in *red* in the following strings with appropriate values from your IAM user account.

   ```
   export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
   export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
   ```

After you complete all prerequisites, copy the following files into a directory in your local environment by using your preferred code editor:

package.json

```
{
  "name": "polygon-rpc",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "ethers": "^6.8.1",
    "@aws-crypto/sha256-js": "^5.2.0",
```

```
    "@aws-sdk/credential-provider-node": "^3.360.0",
    "@aws-sdk/protocol-http": "^3.357.0",
    "@aws-sdk/signature-v4": "^3.357.0",
    "axios": "^1.6.2"
  }
}
```

dispatch-evm-rpc.js

```
const axios = require("axios");
const SHA256 = require("@aws-crypto/sha256-js").Sha256;
const defaultProvider = require("@aws-sdk/credential-provider-node").defaultProvider;
const HttpRequest = require("@aws-sdk/protocol-http").HttpRequest;
const SignatureV4 = require("@aws-sdk/signature-v4").SignatureV4;

// define a signer object with AWS service name, credentials, and region
const signer = new SignatureV4({
  credentials: defaultProvider(),
  service: "managedblockchain",
  region: "us-east-1",
  sha256: SHA256,
});
const rpcRequest = async (rpcEndpoint, rpc) => {

  // parse the URL into its component parts (e.g. host, path)
  let url = new URL(rpcEndpoint);

  // create an HTTP Request object
  const req = new HttpRequest({
    hostname: url.hostname.toString(),
    path: url.pathname.toString(),
    body: JSON.stringify(rpc),
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Accept-Encoding": "gzip",
      host: url.hostname,
    },
  });

  // use AWS SignatureV4 utility to sign the request, extract headers and body
  const signedRequest = await signer.sign(req, { signingDate: new Date() });
```

```
  try {
    //make the request using axios
    const response = await axios({
      ...signedRequest,
      url: url,
      data: req.body,
    });
    return response.data;
  } catch (error) {
    console.error("Something went wrong: ", error);
  }
};

module.exports = { rpcRequest: rpcRequest };
```

sendTx.js

Warning

The following code uses a hardcoded private key to generate a wallet Signer using `Ethers.js` for the sake of demonstration only. Do not use this code in production environments, as it has real funds and poses a security risk.
If needed, contact your account team to advise on wallet and Signer best practices.

```
const ethers = require("ethers");

//set AMB Access Polygon endpoint using token based access (TBA)
let token = "your-billing-token"
let url = `https://mainnet.polygon.managedblockchain.us-east-1.amazonaws.com?
billingtoken=${token}`;

//prevent batch RPCs
let options = {
  batchMaxCount: 1,
};

//create JSON RPC provider with AMB Access endpoint and options
let provider = new ethers.JsonRpcProvider(url, null, options);
```

```
let sendTx = async (to) => {
  //create an instance of the Wallet class with a private key
  //DO NOT USE A WALLET YOU USE ON MAINNET, NEVER USE A RAW PRIVATE KEY IN PROD
  let pk = "wallet-private-key";
  let signer = new ethers.Wallet(pk, provider);

  //use this wallet to send a transaction of POL from one address to another
  const tx = await signer.sendTransaction({
    to: to,
    value: ethers.parseUnits("0.0001", "ether"),
  });

  console.log(tx);
};

sendTx("recipent-address");
```

readTx.js

```
let rpcRequest = require("./dispatch-evm-rpc").rpcRequest;
let ethers = require("ethers");

let getTxDetails = async (txHash) => {
  //set url to a Signature Version 4 endpoint for AMB Access
  let url = "https://mainnet.polygon.managedblockchain.us-east-1.amazonaws.com";

  //set RPC request body to get transaction details
  let getTransactionByHash = {
    id: "1",
    jsonrpc: "2.0",
    method: "eth_getTransactionByHash",
    params: [txHash],
  };

  //make RPC request for transaction details
  let txDetails = await rpcRequest(url, getTransactionByHash);

  //set RPC request body to get recipient user balance
  let getBalance = {
    id: "2",
    jsonrpc: "2.0",
    method: "eth_getBalance",
    params: [txDetails.result.to, "latest"],
```

```
  };

  //make RPC request for recipient user balance
  let recipientBalance = await rpcRequest(url, getBalance);

  console.log("TX DETAILS: ", txDetails.result, "BALANCE: ",
 ethers.formatEther(recipientBalance.result));
};

getTxDetails("your-transaction-id");
```

Once these files are saved to your directory, install the dependencies that are required to run the code using the following command:

```
npm install
```

Send a transaction in Node.js

The preceding example sends the native Polygon Mainnet token (POL) from one address to another by signing a transaction and broadcasting it to the Polygon Mainnet using AMB Access Polygon. To do this, use the `sendTx.js` script, which uses `Ethers.js`, a popular library for interacting with Ethereum and Ethereum-compatible blockchains like Polygon. You need to replace three variables in the code where *highlighted in red*, including the `billingToken` for your *Accessor* token for token based access, the *private key* with which you sign the transaction, and the *recipient's address* that receives the POL.

> **Tip**
>
> We recommended that you create a fresh private key (wallet) for this purpose rather than reusing an existing wallet to eliminate the risk of losing funds. You can use the Ethers library's Wallet class method createRandom() to generate a wallet to test with. Additionally, if you need to request POL from the Polygon Mainnet, you can use the public POL faucet to request a small amount to use for testing.

Once you have your `billingToken`, a funded wallet's *private key*, and the recipient's address added to the code, you run the following code to sign a transaction for *.0001* POL

to be sent from your address to another and broadcast it to Polygon Mainnet invoking the `eth_sendRawTransaction` JSON-RPC using the AMB Access Polygon.

```
node sendTx.js
```

The response received back resembles the following:

```
TransactionResponse {
provider: JsonRpcProvider {},
blockNumber: null,
blockHash: null,
index: undefined,
hash: '0x8d7538b4841261c5120c0a4dd66359e8ee189e7d1d34ac646a1d9923********',
type: 2,
to: '0xd2bb4f4f1BdC4CB54f715C249Fc5a991********',
from: '0xcf2C679AC6cb7de09Bf6BB6042ecCF05********',
nonce: 2,
gasLimit: 21000n,
gasPrice: undefined,
maxPriorityFeePerGas: 16569518669n,
maxFeePerGas: 16569518685n,
data: '0x',
value: 100000000000000n,
chainId: 80001n,
signature: Signature {
r: "0x1b90ad9e9e4e005904562d50e904f9db10430a18b45931c059960ede337238ee",
s: "0x7df3c930a964fd07fed4a59f60b4ee896ffc7df4ea41b0facfe82b470db448b7",
yParity: 0,
networkV: null
},
accessList: []
}
```

The response constitutes the transaction receipt. Save the value of the property `hash`. This is the identifier for the transaction you just submitted to the blockchain. You use this property in the read transaction example to get additional details about this transaction from the Polygon Mainnet.

Note that the `blockNumber` and `blockHash` are `null` in the response. This is because the transaction has not yet been recorded in a block on the Polygon network. Note that these values are defined later and you might see them when you request the transaction details in the following section.

Read a transaction in Node.js

In this section, you request the transaction details for the previously submitted transaction and retrieve the POL balance for the recipient address using read requests to the Polygon Mainnet using AMB Access Polygon. In the `readTx.js` file, replace the variable labeled *`your-transaction-id`* with the `hash` you saved from the response from running the code in the previous section.

This code uses a utility, `dispatch-evm-rpc.js`, which signs HTTPS requests to AMB Access Polygon with the requisite Signature Version 4 (SigV4) modules from the AWS SDK and sends requests using the widely used HTTP client, AXIOS.

The response received back resembles the following:

```
TX DETAILS: {
blockHash: '0x59433e0096c783acab0659175460bb3c919545ac14e737d7465b3ddc********',
blockNumber: '0x28b4059',
from: '0xcf2c679ac6cb7de09bf6bb6042eccf05b7fa1394',
gas: '0x5208',
gasPrice: '0x3db9eca5d',
maxPriorityFeePerGas: '0x3db9eca4d',
maxFeePerGas: '0x3db9eca5d',
hash: '0x8d7538b4841261c5120c0a4dd66359e8ee189e7d1d34ac646a1d9923********',
input: '0x',
nonce: '0x2',
to: '0xd2bb4f4f1bdc4cb54f715c249fc5a991********',
transactionIndex: '0x0',
value: '0x5af3107a4000',
type: '0x2',
accessList: [],
chainId: '0x13881',
v: '0x0',
r: '0x1b90ad9e9e4e005904562d50e904f9db10430a18b45931c059960ede337238ee',
s: '0x7df3c930a964fd07fed4a59f60b4ee896ffc7df4ea41b0facfe82b470db448b7'
} BALANCE: 0.0003
```

The response represents the transaction details. Note that the `blockHash` and `blockNumber` are now likely defined. This indicates that the transaction has been recorded in a block. If these values are still `null`, wait a few minutes, then run the code again to check if your transaction has been included in a block. Lastly, the hexadecimal representation of the recipient address balance

(0x110d9316ec000) is converted to decimal using Ethers' `formatEther()` method, which converts the hex to decimal and shifts decimal places by *18 (10^18)* to give the true balance in POL.

Tip

While the preceding code examples illustrate how to use Node.js, Ethers, and Axios to utilize a few of the supported JSON-RPCs on AMB Access Polygon, you can modify the examples and write other code to build your applications on Polygon using this service. For a full list of supported JSON-RPCs on AMB Access Polygon, see Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon.

Creating and managing Accessor tokens for token-based access to make AMB Access Polygon requests

You can also use *Accessor* tokens to make JSON-RPC calls to the Polygon network endpoints as a convenient alternative to the Signature Version 4 (SigV4) signing process. You must provide a `BILLING_TOKEN` from one of the Accessor tokens you create and add as a parameter with your calls.

Important

- If you prioritize security and auditability over convenience, use the SigV4 signing process instead.
- You can access the Polygon JSON-RPCs using Signature Version 4 (SigV4) and token-based access. However, if you choose to use both protocols, your request is rejected.
- You must never embed Accessor tokens in user-facing applications.

In the console, the **Token Accessors** page displays a list of all the Accessor tokens that you can use to make AMB Access Polygon JSON-RPC calls from your AWS account from code on a client.

For more information about AMB Access Polygon JSON-RPC requests, see Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon.

You can create and manage Accessor tokens using the AWS Management Console. You can also create and manage Accessor tokens using the following API operations: `CreateAccessor`, `GetAccessor`, `ListAccessors`, and `DeleteAccessor`. A `BILLING_TOKEN` is a property of the Accessor. This `BILLING_TOKEN` property is used to track your Accessor and for billing AMB Access Polygon JSON-RPC requests made from your AWS account.

All API actions related to creating and managing Accessor tokens are also available through the AWS Management Console, AWS CLI, and SDKs.

Creating an Accessor token for token-based access

You can create an Accessor token and use it to make AMB Access Polygon API calls on any AMB Access Polygon node in your AWS account.

Create an Accessor token to make AMB Access Polygon JSON-RPC requests using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Token Accessors**.
3. Choose **Create Accessor**.
4. Choose a valid *Polygon* blockchain **Network**.
5. Optional, add **Tags** for your Accessor.
6. Choose **Create Accessor** to create a new Accessor token.

Create an Accessor token to make AMB Access Polygon JSON-RPC requests using the AWS CLI

```
aws managedblockchain create-accessor --accessor-type BILLING_TOKEN --network-type
 POLYGON_MAINNET
```

The previous command returns the `AccessorId` along with the `BillingToken`, as shown in the following example.

```
{
"AccessorId": "ac-NGQ6QNKXLNEBXD3UI6********",
"NetworkType": "POLYGON_MAINNET",
"BillingToken": "jZlP8OUI-PcQSKINyX9euJJDC5-IcW9e-n********"
}
```

The *key* element in your response is the `BillingToken`. You can use this property to make AMB Access Polygon JSON-RPC calls. Some values in the example have been obfuscated for security reasons but will appear fully in actual responses.

> **Note**
>
> After the operation is run, Managed Blockchain provisions and configures the token for you. The length of this process depends on many variables.

Viewing an Accessor token details

You can view the properties for each Accessor token that your AWS account owns. For example, you can view the Accessor ID or the Amazon Resource Name (ARN) of the Accessor. You can also view the status, the type, the creation date, and the `BillingToken`.

To view an Accessor token's information using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. In the navigation pane, choose **Token Accessors**.
3. Choose the **Accessor ID** of the token from the list.

The token details page the pops up. From this page, you can view the properties of the token.

To view an Accessor token's information using the AWS CLI

Run the following command to view the details of an Accessor token. Replace values of `--accessor-id` with your Accessor ID.

```
aws managedblockchain get-accessor --accessor-id ac-NGQ6QNKXLNEBXD3UI6********
```

The `BillingToken` and other key properties are returned as shown in the following example. Some values in the example have been obfuscated for security reasons but appear fully in actual responses.

```
{
  "Accessor": {
  "Id": "ac-NGQ6QNKXLNEBXD3UI6********",
  "Type": "BILLING_TOKEN",
  "BillingToken": "jZlP8OUI-PcQSKINyX9euJJDC5-IcW9e-n********",
  "Status": "AVAILABLE",
  "NetworkType": "POLYGON_MAINNET"
  "CreationDate": "2022-01-04T23:09:47.750Z",
  "Arn": "arn:aws:managedblockchain:us-east-1:666666666666:accessors/ac-
NGQ6QNKXLNEBXD3UI6********"
  }
}
```

Deleting an Accessor token

When you delete an Accessor token, the token changes from the `AVAILABLE` to the `PENDING_DELETION` status. You can't use an Accessor token with the `PENDING_DELETION` status.

To delete an Accessor token using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. In the navigation pane, choose **Token Accessors**.
3. Select the Accessor token that you want from the list.
4. Choose **Delete**.
5. Confirm your choice.

You're returned to the **Tokens accessors** page with your deleted Accessor token. The page displays the `PENDING_DELETION` status.

To delete an Accessor token using the AWS CLI

The following example shows how to delete a token. Use the `delete-accessor` command to delete a token. Set the value of `--accessor-id` with your Accessor ID.

Deleting an Accessor token using the AWS CLI

```
aws managedblockchain delete-accessor --accessor-id ac-NGQ6QNKXLNEBXD3UI6********
```

If this command runs successfully, no messages are returned.

Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon

Amazon Managed Blockchain provides API operations for creating and managing token accessors for AMB Access Polygon. For more information, see the *Managed Blockchain API Reference Guide* .

The following topic provides a list and reference of the Polygon JSON-RPCs that AMB Access Polygon supports. Each supported JSON-RPC has a brief description of its use. You use the Polygon JSON-RPCs to query and get smart contract data, get transaction details, submit transactions, and other utilities such as running traces on transactions, and estimate fees.

AMB Access Polygon supports the following JSON-RPC methods. Each supported JSON-RPC has a category and a brief description of its utility and its default request quotas. Unique considerations for using the JSON-RPC method with Amazon Managed Blockchain are indicated where applicable.

Note

- Any methods that aren't listed are not supported.
- When making calls to the Polygon JSON-RPCs on Amazon Managed Blockchain, you can do so over an HTTPS connection authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account can make Polygon JSON-RPC calls. To do this, AWS credentials (an access key ID and a secret access key) must be provided with the call.
- You can also use token-based access as a convenient alternative to the Signature Version 4 (SigV4) signing process. If you prioritize security and auditability over convenience, use the SigV4 signing process instead. However, if you use both SigV4 and token-based access, your requests will not work.
- JSON-RPC batch requests aren't supported on Amazon Managed Blockchain (AMB) Access Polygon for this preview.
- The **Quotas** column in the following table lists the quota for each JSON-RPC. Quotas are set in requests per second (RPS) per Region per Polygon network (Mainnet) for each JSON-RPC.

  For increasing your quota, you must contact Support. To contact Support, sign into the AWS Support Center Console. Choose **Create case**. Choose **Technical**. Choose *Managed*

Blockchain as your **service**. Choose *Access:Polygon* as your **Category** and *General guidance* as your **Severity**. Enter *RPC Quota* as the **Subject** and in the **Description** text box list the JSON-RPC and the quota limits applicable to your needs in *RPS per Polygon network per Region*. **Submit** your case.

Category	JSON-RPC	Description	Considerations
Ethereum	eth_blockNumber	Returns the number of the most recent block.	
	eth_call	Immediately runs a new message call without creating a transaction on the blockchain.	`eth_call` consumes 0 gas, but has a gas parameter for messages that require it.
	eth_chainId	Returns an integer value for the currently configure d `Chain Id` value that's introduced in EIP-155. Returns `None` if no `Chain Id` is available.	
	eth_estimateGas	Estimates and returns the gas that's required for a transaction without adding the transaction to the blockchain.	

Category	JSON-RPC	Description	Considerations
	eth_feeHistory	Returns a collection of historical gas information.	
	eth_gasPrice	Returns the current price per gas in Wei.	
	eth_getBalance	Returns the balance of an account for the specified account address and block identifier.	
	eth_getBlockByHash	Returns information about the block specified using the block hash.	
	eth_getBlockByNumber	Returns information about the block specified using the block number.	
	eth_getBlockReceipts	Returns receipts about the block specified using the block number.	
	eth_getBlockTransactionCountByHash	Returns the number of transactions in the block specified using the block hash.	

Category	JSON-RPC	Description	Considerations
	eth_getBlockTransactionCountByNumber	Returns the number of transactions in the block specified using the block number.	
	eth_getCode	Returns the code at the specified account address and block identifier.	
	eth_getLogs	Returns an array of all logs for a specified filter object.	You can make `eth_getloqs` requests on *any* block range with a 1K block range by default when a contract address is provided. Contracts with high activity may be limited to smaller block ranges. If no contract address is provided, the block range will be *8*.

Category	JSON-RPC	Description	Considerations
	eth_getRawTransactionByHash	Returns the raw form of the transacti on specified by the `transacti on_hash` .	
	eth_getStorageAt	Returns the value of the specified storage position for the specified account address and block identifie r.	
	eth_getTransactionByBlockHashAndInde x	Returns informati on about a transaction using the specified block hash and transacti on index position.	
	eth_getTransactionByBlockNu mberAndIndex	Returns informati on about a transaction using the specified block number and transaction index position.	
	eth_getTransactionByHash	Returns informati on about the transaction with the specified transaction hash.	

Category	JSON-RPC	Description	Considerations
	eth_getTransactionCount	Returns the number of transactions sent from the specified address and block identifier.	
	eth_getTransactionReceipt	Returns the receipt of the transaction using the specified transaction hash.	
	eth_getUncleByBlockHashAndIndex	Returns informati on about the uncle block specified using the block hash and uncle index position.	
	eth_getUncleByBlockNumberAndIndex	Returns informati on about the uncle block specified using the block number and uncle index position.	
	eth_getUncleCountByBlockHash	Returns the number of counts in the uncle specified using the uncle hash.	

Category	JSON-RPC	Description	Considerations
	eth_getUncleCountByBlockNumber	Returns the number of counts in the uncle specified using the uncle number.	
	eth_maxPriorityFeePerGas	Returns the fee per gas that's an estimate of how much you can pay as a priority fee, or "tip," to get a transaction included in the current block.	Generally you use the value that's returned from this method to set the `maxFeePerGas` in the subsequen t transaction that you're submitting.
	eth_protocolVersion	Returns the current Ethereum protocol version.	
	eth_sendRawTransaction	Creates a new message call transaction or a contract creation for signed transactions.	Managed Blockchain supports raw transactions only. You must create and sign transacti ons before sending them.

Category	JSON-RPC	Description	Considerations
Debug	debug_traceBlockByHash	Returns the possible tracing result number by executing all transactions in the block specified by the block hash with a tracer (Trace Mode required).	
	debug_traceBlockByNumber	Returns the tracing result by executing all transactions in the block specified by number with a tracer (Trace Mode required).	
	debug_traceCall	Returns the number of possible tracing results by executing an eth call within the context of the given block execution (Trace Mode required).	
	debug_traceTransaction	Returns all traces of a given transacti on (Trace Mode required).	

Category	JSON-RPC	Description	Considerations
Net	net_version	Returns the current network id.	
Trace	trace_block	Returns a full stack trace of all invoked opcodes of all transactions that were included in a block.	
	trace_call	Returns the number of possible tracing results by executing an eth call within the context of the given block execution (Trace Mode required).	
	trace_transaction	Returns all traces of a given transacti on (Trace Mode required).	
Tx Pool	txpool_content	Returns all pending and queued transacti ons.	

Category	JSON-RPC	Description	Considerations
	txpool_status	Provides a count of all transactions currently pending inclusion in the next blocks, and those that are queued (being scheduled for future execution only).	
Web	web3_clientVersion	Returns the current client version.	

Polygon use cases with Amazon Managed Blockchain (AMB) Access Polygon

The Polygon blockchain is commonly used in building decentralized applications (dApps) related to NFTs, Web3 games, and tokenization use cases, among others. This topic provides a list of some of the use cases that you can implement using Amazon Managed Blockchain (AMB) Access Polygon.

Topics

Analyze Polygon NFT data

You can collect data about Polygon NFTs, including information like transfer events and NFT metadata for a specified period. You can then analyze this data to draw insights like which NFTs are trending or which users are most frequently interacting with a given collection.

For more information, see Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon.

Support NFT purchases

You can use AMB Access Polygon to submit transactions for NFT purchases using initial mint, allowlists, or on the secondary market. Using a combination of other AWS services, you can then permit purchases using credit cards, accepting Fiat or cryptocurrencies, with a quick settlement for all stakeholders involved.

For more information, see Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon.

Create a Polygon wallet

You can use AMB Access Polygon to serve critical functions of digital asset wallets, such as reading user token balances from smart contracts on the blockchain or broadcasting signed transactions to the blockchain.

For more information, see Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon.

Wallet as a service

You can use AMB Access Polygon to develop an operating wallet-as-a-service needed to support common wallet transactions such as checking a balance, asset transfer, asset send, and fee estimations, using the supported Polygon JSON-RPCs.

For more information, see Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon.

Token-gated experiences

You can use AMB Access Polygon to build token-gated experiences for your users. For example, you can conditionally provide access to a piece of content only to the owners of a specific NFT. To achieve this, you must read the blockchain to determine the NFT ownership of a user's address.

For more information, see Managed Blockchain API and the JSON-RPCs supported with AMB Access Polygon.

Tutorials for Amazon Managed Blockchain (AMB) Access Polygon

The following tutorials highlighted in this section are *Community Articles* from AWS re:Post that provide walkthroughs to help you learn how to perform some common tasks on the Polygon blockchain using AMB Access Polygon.

- Sending transactions using AMB Access Polygon and web3.js
- Deploy a smart contract using AMB Access Polygon and Hardhat Ignition
- Interacting with a smart contract
- Retrieve current price data off-chain using AMB Access Polygon and Chainlink data feeds
- Analyze ERC-20 token data on Polygon Mainnet with AMB Access

Security in Amazon Managed Blockchain (AMB) Access Polygon

Cloud security at AWS is of the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as both security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Managed Blockchain (AMB) Access Polygon, see AWS Services in Scope by Compliance Program.
- **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 company's requirements, and applicable laws and regulations.

To provide data protection, authentication, and access control, Amazon Managed Blockchain uses AWS features and the features of the open-source framework running in Managed Blockchain.

This documentation helps you understand how to apply the shared responsibility model when using AMB Access Polygon. The following topics show you how to configure AMB Access Polygon to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AMB Access Polygon resources.

Topics

Data protection in Amazon Managed Blockchain (AMB) Access Polygon

The AWS shared responsibility model applies to data protection in Amazon Managed Blockchain (AMB) Access Polygon. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AMB Access Polygon or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption

Data encryption helps prevent unauthorized users from reading data from a blockchain network and the associated data storage systems. This includes data that might be intercepted as it travels the network, known as *data in transit*.

Encryption in transit

By default, Managed Blockchain uses an HTTPS/TLS connection to encrypt all the data that's transmitted from a client computer that runs the AWS CLI to AWS service endpoints.

You don't need to do anything to enable the use of HTTPS/TLS. It's always enabled unless you explicitly disable it for an individual AWS CLI command by using the `--no-verify-ssl` command.

Identity and access management for Amazon Managed Blockchain (AMB) Access Polygon

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AMB Access Polygon resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AMB Access Polygon.

Service user – If you use the AMB Access Polygon service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AMB Access

Polygon features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AMB Access Polygon, see Troubleshooting Amazon Managed Blockchain (AMB) Access Polygon identity and access.

Service administrator – If you're in charge of AMB Access Polygon resources at your company, you probably have full access to AMB Access Polygon. It's your job to determine which AMB Access Polygon features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AMB Access Polygon, see How Amazon Managed Blockchain (AMB) Access Polygon works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AMB Access Polygon. To view example AMB Access Polygon identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Polygon.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or

store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's

permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Managed Blockchain (AMB) Access Polygon works with IAM

Before you use IAM to manage access to AMB Access Polygon, learn what IAM features are available to use with AMB Access Polygon.

IAM features you can use with Amazon Managed Blockchain (AMB) Access Polygon

IAM feature	AMB Access Polygon support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	No
Policy condition keys	No
ACLs	No
ABAC (tags in policies)	No
Temporary credentials	No
Principal permissions	No
Service roles	No

IAM feature	AMB Access Polygon support
Service-linked roles	No

To get a high-level view of how AMB Access Polygon and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AMB Access Polygon

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AMB Access Polygon

To view examples of AMB Access Polygon identity-based policies, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Polygon.

Resource-based policies within AMB Access Polygon

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AMB Access Polygon

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AMB Access Polygon actions, see Actions Defined by Amazon Managed Blockchain (AMB) Access Polygon in the *Service Authorization Reference*.

Policy actions in AMB Access Polygon use the following prefix before the action:

```
managedblockchain:
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "managedblockchain::action1",
      "managedblockchain::action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `InvokeRpcPolygon`, include the following action:

```
"Action": "managedblockchain::InvokeRpcPolygon*"
```

To view examples of AMB Access Polygon identity-based policies, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Polygon.

Policy resources for AMB Access Polygon

Supports policy resources: No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AMB Access Polygon resource types and their ARNs, see Resources Defined by Amazon Managed Blockchain (AMB) Access Polygon in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Managed Blockchain (AMB) Access Polygon .

To view examples of AMB Access Polygon identity-based policies, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Polygon.

Policy condition keys for AMB Access Polygon

Supports service-specific policy condition keys: No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AMB Access Polygon condition keys, see Condition Keys for Amazon Managed Blockchain (AMB) Access Polygon in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon Managed Blockchain (AMB) Access Polygon .

To view examples of AMB Access Polygon identity-based policies, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Polygon.

ACLs in AMB Access Polygon

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with AMB Access Polygon

Supports ABAC (tags in policies): No

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then

you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with AMB Access Polygon

Supports temporary credentials: No

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for AMB Access Polygon

Supports forward access sessions (FAS): No

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for AMB Access Polygon

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break AMB Access Polygon functionality. Edit service roles only when AMB Access Polygon provides guidance to do so.

Service-linked roles for AMB Access Polygon

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Polygon

By default, users and roles don't have permission to create or modify AMB Access Polygon resources. They also can't perform tasks by using the AWS Management Console, AWS Command

Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AMB Access Polygon, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon Managed Blockchain (AMB) Access Polygon in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the AMB Access Polygon console
- Allow users to view their own permissions
- Accessing Polygon networks

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AMB Access Polygon resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to

specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AMB Access Polygon console

To access the Amazon Managed Blockchain (AMB) Access Polygon console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AMB Access Polygon resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AMB Access Polygon console, also attach the AMB Access Polygon `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Accessing Polygon networks

> **Note**
>
> In order to access the public endpoints for the Polygon `mainnet` and `mainnet` to make JSON-RPC calls, you will need user credentials (`AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`) that have the appropriate IAM permissions for AMB Access Polygon.

Example IAM Policy to access all Polygon Networks

This example grants an IAM user in your AWS account access to all Polygon networks.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessAllPolygonNetworks",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:InvokeRpcPolygon*"
            ],
            "Resource": "*"
        }
    ]
}
```

Example IAM Policy to access the Polygon Mainnet network

This example grants an IAM user in your AWS account access to the Polygon Mainnet network.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessPolygonTestnet",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:InvokeRpcPolygonMainnet"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting Amazon Managed Blockchain (AMB) Access Polygon identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AMB Access Polygon and IAM.

Topics

I am not authorized to perform an action in AMB Access Polygon

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `managedblockchain::`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 managedblockchain::GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `managedblockchain::`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AMB Access Polygon.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AMB Access Polygon. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AMB Access Polygon resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AMB Access Polygon supports these features, see How Amazon Managed Blockchain (AMB) Access Polygon works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Logging Amazon Managed Blockchain (AMB) Access Polygon events by using AWS CloudTrail

> **Note**
>
> Amazon Managed Blockchain (AMB) Access Polygon doesn't support management events.

Amazon Managed Blockchain runs on AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Managed Blockchain. CloudTrail captures who invoked the AMB Access Polygon endpoints for Managed Blockchain as data plane events.

If you create a properly configured trail that is subscribed to receive the desired data plane events, you can receive continuous delivery of AMB Access Polygon related CloudTrail events to an S3 bucket. Using the information that's collected by CloudTrail, you can determine that a request was made to one of the AMB Access Polygon endpoints, the IP address that the request came from, who made the request, when it was made, and other additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AMB Access Polygon information in CloudTrail

CloudTrail is enabled on your AWS account when you create it. However, you must configure the data plane events to view who invoked the AMB Access Polygon endpoints.

For an ongoing record of events in your AWS account, including events for AMB Access Polygon, create a trail. A *trail* enables CloudTrail to deliver log files to an S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all supported Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. Additionally, you can configure other AWS services to analyze further and act on the event data collected in CloudTrail logs. For more information, see the following:

- Using CloudTrail to track Polygon JSON-RPCs
- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail

- Receiving CloudTrail log files from multiple Regions and Receiving CloudTrail log files from multiple accounts

By analyzing the CloudTrail data events, you can monitor who invoked the AMB Access Polygon endpoints.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials
- Whether the request was made with temporary security credentials for a role or a federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity element.

Understanding AMB Access Polygon log file entries

For data plane events, a trail is a configuration that enables delivery of events as log files to a specified S3 bucket. Each CloudTrail log file contains one or more log entries that represent a single request from any source. These entries provide details about the requested action, including the date and time of the action, and any associated request parameters.

> **Note**
>
> CloudTrail data events in the log files aren't an ordered stack trace of the AMB Access Polygon API calls, so they don't appear in any specific order.

Using CloudTrail to track Polygon JSON-RPCs

You can use CloudTrail to track who in your account invoked the AMB Access Polygon endpoints and which JSON-RPC was invoked as *data events*. By default, when you create a trail, data events aren't logged. To record who invoked the AMB Access Polygon endpoints as CloudTrail data events, you must explicitly add the supported resources or resource types for which you want to collect activity to a trail. AMB Access Polygon supports adding data events by using the AWS Management

Console, AWS CLI, and SDK. For more information, see Log events by using advanced selectors in the *AWS CloudTrail User Guide* .

To log data events in a trail, use the put-event-selectors operation after you create the trail. Use the `--advanced-event-selectors` option to specify the `AWS::ManagedBlockchain::Network` resource types in order to start logging data events to determine who invoked the AMB Access Polygon endpoints.

Example Data event log entry of all your account's AMB Access Polygon endpoints requests

The following example demonstrates how to use the `put-event-selectors` operation to log all your account's AMB Access Polygon endpoint requests for the trail `my-polygon-trail` in the `us-east-1` Region.

```
aws cloudtrail put-event-selectors \

--region us-east-1 \
--trail-name my-polygon-trail  \
--advanced-event-selectors '[{
    "Name": "Test",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::ManagedBlockchain::Network"] } ]}]'
```

After you subscribe, you can track usage in the S3 bucket that is connected to the trail specified in the previous example.

The following result shows a CloudTrail data event log entry of the information that's collected by CloudTrail. You can determine that a Polygon JSON-RPC request was made to one of the AMB Access Polygon endpoints, the IP address that the request came from, who made the request, when it was made, and other additional details. Some values in the following example have been obfuscated for security reasons but appear fully in actual log entries.

```
{
        "eventVersion": "1.09",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "AROA554UO62RJ7KSB7FAX:777777777777",
            "arn": "arn:aws:sts::111122223333:assumed-role/Admin/777777777777",
            "accountId": "111122223333"
        },
```

```
        "eventTime": "2023-04-12T19:00:22Z",
        "eventSource": "managedblockchain.amazonaws.com",
        "eventName": "gettxout",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "111.222.333.444",
        "userAgent": "python-requests/2.28.1",
        "errorCode": "-",
        "errorMessage": "-",
        "requestParameters": {
            "jsonrpc": "2.0",
            "method": "gettxout",
            "params": [],
            "id": 1
        },
        "responseElements": null,
        "requestID": "DRznHHEj********",
        "eventID": "baeb232d-2c6b-46cd-992c-0e40********",
        "readOnly": true,
        "resources": [{
            "type": "AWS::ManagedBlockchain::Network",
            "ARN": "arn:aws:managedblockchain:::networks/n-polygon-mainnet"
        }],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "111122223333",
        "eventCategory": "Data"
}
```

Document history for the AMB Access Polygon User Guide

The following table describes the documentation releases for AMB Access Polygon.

Change	Description	Date
Updated quotas for JSON-RPC	The quotas that AMB Access Polygon supports for each supported JSON-RPC are updated.	April 12, 2024
End of support for the Mumbai testnet network	AMB Access Polygon ended support of the Mumbai testnet on April 15, 2024.	April 10, 2024
Addition of the Tutorials topic	AMB Access Polygon tutorials from the *Community Articles* section of AWS re:Post.	April 9, 2024
Public preview	Public preview release of the Amazon Managed Blockchain (AMB) Access Polygon service.	November 24, 2023