MCP policy reference

- Evaluation model
- Actions
- Resources
- Resource attributes
  - Server type values
  - Server identity values
- Context fields
- Approval annotation
- Limitations

MCP policies are organization policies written in Cedar using Docker's `MCP` namespace. This reference defines the Docker-specific policy surface for Model Context Protocol (MCP) activity made available to sandboxes through Docker's MCP gateway.

Use this reference with MCP access policies for common policy patterns. For the Cedar language, see the Cedar documentation.

Evaluation model

Cedar evaluates MCP requests against a principal, action, resource, and context. For Docker MCP policies, policy scope supplies the principal. Write policies against the action, resource, and context. Clauses that reference principal attributes, such as `principal in ...`, `principal.role`, or `principal.tenant`, don't match.

Governed MCP activity is default deny. A request is blocked unless a matching `permit` allows it. A matching `forbid` overrides any `permit`, including a permit annotated with `@requireApproval`.

For details about when Docker Sandboxes evaluates MCP policies for a user, see Govern the server lifecycle.

An actionless `permit` matches every MCP action that reaches Cedar evaluation:

```
permit (principal, action, resource);
```

Actions

Action	Governs	Notes
`register`	MCP server registration	Server registration needs an explicit `permit`. Use server attributes to scope registration.
`invokeTool`	MCP tool calls	Most tool access policies target this action.
`invokePrimordial`	Gateway meta-tool calls	Applies to built-in gateway tools such as `mcp-exec`, `mcp-add`, `code-mode`, and OAuth authorization helpers.
`readResource`	MCP resource reads	Rules match `MCP::Resource` and `resource.uri`.
`getPrompt`	MCP prompt retrieval	Rules match `MCP::Prompt` and `resource.name`.
`listTools`	MCP tool listing	Defined in the schema but not Cedar-gated. Tool listings can include tools denied at invocation.
`listResources`	MCP resource listing	Defined in the schema but not Cedar-gated. Resource listings can include resources denied by policy.
`subscribeResource`	MCP resource subscription	Defined in the schema but not Cedar-gated.

Resources

Match resources with the MCP entity type and attributes for the request.

Entity	Match with	Notes
`MCP::Server`	Registered server name	The canonical server identity is the `resource.identityURL` attribute, not the entity ID.
`MCP::Tool`	Bare tool name	Use `resource.name`. Display prefixes aren't included. A bare-name match applies to every server exposing a tool with that name.
`MCP::Resource`	Resource URI	Use `resource.uri`.

Entity	Match with	Notes
`MCP::Prompt`	Prompt name	Use `resource.name`.
`MCP::Primordial`	Gateway meta-tool name	Match a specific primordial with an entity reference.

Examples:

```
resource in MCP::Server::"notion"
resource.name == "move_file"
resource.uri like "*/docs/*"
resource in MCP::Primordial::"code-mode"
```

Resource attributes

Tool annotation attributes come from MCP tool annotations or catalog metadata and are advisory.

Attribute	Applies to	Notes
`resource.name`	Tools and prompts	For tools, this is the bare tool name, not a display-prefixed name.
`resource.uri`	Resources	Use with string operators such as `like`.
`resource.readOnly`	Tools	Defaults to `false` when a tool doesn't declare it.
`resource.destructive`	Tools	Defaults to `true` when a tool doesn't declare it.
`resource.idempotent`	Tools	Defaults to `false` when a tool doesn't declare it.
`resource.openWorld`	Tools	Defaults to `true` when a tool doesn't declare it.
`resource.type`	Servers	Use for server registration rules. See Server type values.
`resource.identityURL`	Servers	Canonical server identity. The value depends on the registration type. See Server identity values.
`resource.requiresOAuth`	Servers	Use for server registration rules.
`resource.requiresNetwork`	Servers	Use for server registration rules.

Attribute	Applies to	Notes
`resource.command`	Servers	Local stdio server command, such as `npx` or `docker`, when available. Empty for remote servers and registrations that don't include command details.
`resource.args`	Servers	Local stdio server arguments when available. This is a set, so `.contains()` can match values. Empty when no command details are available.

Use `like` for string attributes. In Cedar, `like` uses `*` as its wildcard, matches the full string, treats `?` as a literal character, and treats `\*` as a literal asterisk.

Use `.contains()` only on set attributes, such as `resource.args`. On string attributes, use `like`.

Server type values

Local gateway registrations use these values:

- `local-stdio`: a host-run stdio server. This includes explicit commands and OCI-packaged stdio servers resolved from metadata with `--local`.
- `container-stdio`: an OCI-packaged server resolved from metadata without `--local`. This value can appear in `register` decisions, but the local gateway can't attach or run this server type.
- `remote-dcr`: a remote endpoint that doesn't require OAuth or supports OAuth Dynamic Client Registration.
- `remote-no-dcr`: a remote OAuth endpoint that doesn't support Dynamic Client Registration.

Server identity values

For a remote server, `resource.identityURL` is the endpoint URL. For an explicit local command, it is the resolved executable path on the host. For a `--local` metadata registration, it is `local://stdio/<name>`, not the registry or manifest URL.

Context fields

Field	Notes

`context.request_time`	Bound for tool calls, built-in gateway tool calls, resource reads, and prompt retrieval. Registration requests don't include it.
`context.args`	Arguments for `invokeTool` and `invokePrimordial` requests evaluated by the gateway. Present when arguments are available as a JSON object.

A registration `permit` conditioned on `context.request_time` doesn't match, so the registration falls to default deny.

Guard tool-call argument rules with `context has args` and a field check:

```
permit (principal, action == MCP::Action::"invokeTool", resource)
when {
  resource.name == "approve_expense" &&
  context has args &&
  context.args has amount &&
  context.args.amount <= 500
};
```

A `permit` gated on missing arguments doesn't match, so the request falls to default deny. A `forbid` gated on missing arguments doesn't match, so it doesn't block the request.

Only object-shaped tool arguments are represented in `context.args`. Unsupported or malformed arguments are omitted.

Approval annotation

Use `@requireApproval("reason")` on a `permit` statement to require in-session confirmation through MCP elicitation before a matching request runs:

```
@requireApproval("write tool call")
permit (principal, action == MCP::Action::"invokeTool", resource)
when { resource.readOnly == false };
```

When a request matches the annotated `permit` and no `forbid` overrides it, the policy engine returns an approval-required outcome. The annotation string is shown as the elicitation reason. An approval-required outcome takes precedence over a normal `permit`. A matching `forbid` denies the request without an elicitation.

For the request flow and trust model, see Require confirmation with MCP elicitation.

Approval requires a client session that can present an MCP elicitation request to the user. If the request can't be presented for approval, the request is denied. Approval is an in-session confirmation, not an out-of-band approval workflow. Each confirmation applies to one authorization request. After the client confirms, the gateway re-evaluates the request with an approval digest.

`sbx mcp add` can't present an elicitation request. A registration permit with `@requireApproval` therefore results in a denial.

Only the exact annotation name `@requireApproval` applies approval behavior. Other annotation names, such as `@requireConsent` or `@requireConfirmation`, don't require approval.

Limitations

- Tool and resource listing actions aren't Cedar-gated. Listings can include entries that a policy denies when the sandbox tries to use them.
- Approval-gated requests are denied when the execution context can't relay an MCP elicitation to the originating client. This includes tool calls made from inside `code-mode`.
- Registration policy is evaluated when a server is registered. It doesn't remove existing registrations or stop an already-loaded server by itself. Govern existing registrations with use-time rules such as `invokeTool`, `readResource`, and `getPrompt`.
- Server command and argument rules using `resource.command` or `resource.args` apply only when the resolved server registration includes local stdio command details. Remote servers and metadata-resolved local servers can have empty values for those attributes. Use `resource.type == "local-stdio"` to match host-run servers independently of command details.
- Principal-based rules don't take effect. Use organization and team policy scope to target users.
- Server groups aren't supported in MCP policy. Reference servers individually.

□□□ #2
□ □□□ □□□ 2026-10-31 19:01:39 UTC
□ □□□ □□□ 2026-10-31 19:01:40 UTC