Jasper - Post AMI Setup Guide

Configuring a new EC2 instance launched from the Jasper-ready AMI

This guide covers the steps needed **after** launching a new EC2 instance from the pre-built Jasper AMI. That AMI already has Ubuntu 24.04, the `jasper` user, and Jasper itself installed.

Note: Three setup scripts (configure_llm.sh, setup_toolset_mcp.sh, teams_slack_bot_setup.sh) write a full log of everything they do to /var/log/jasper/ (e.g. configure_llm_<timestamp>.log, plus a *-latest.log symlink to the most recent run). If a script fails, check that log first — it captures the same output shown on screen, and no secrets are ever written to it. Every ad-hoc holmes CLI invocation is logged the same way to /var/log/jasper/cli/. Each MCP server logs its own tool calls to /var/log/jasper/mcp/<server>.log. Teams/Slack conversations (every message, reply, and tool call) are logged separately under /home/jasper/jasper-deploy/holmesgpt/logs/<date>/, since those services run from that directory.

Step 1 — Create and Attach IAM Instance Role (Prerequisite)

Create an IAM instance role (i.e. Role Name: `Jasper_Access_Role`). **No static AWS credentials should ever be configured** — do not run `aws configure`. Every script and custom toolset relies exclusively on this instance role for AWS authentication.

Required permissions:

Option 1 — Least-privilege (recommended) — a custom policy scoped exactly to what Jasper's toolsets use today. Attach as an inline or customer-managed policy on `Jasper_Access_Role`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "JasperSSM",
      "Effect": "Allow",
      "Action": ["ssm:GetParameter", "ssm:GetParametersByPath", "ssm:PutParameter"],
      "Resource": [
        "arn:aws:ssm:*:*:parameter/jasper/config",
        "arn:aws:ssm:*:*:parameter/jasper/config/*"
      ]
    },
    {
      "Sid": "JasperKMS",
      "Effect": "Allow",
      "Action": ["kms:Decrypt", "kms:Encrypt"],
      "Resource": "*"
    },
    {
      "Sid": "JasperS3",
      "Effect": "Allow",
      "Action": ["s3:ListBucket", "s3:GetObject"],
      "Resource": [
        "arn:aws:s3:::rlcatalyst-jasper-toolsets",
        "arn:aws:s3:::rlcatalyst-jasper-toolsets/Toolsets/*"
      ]
    },
    {
```

```
      "Sid": "JasperAWSReadOnly",
      "Effect": "Allow",
      "Action": [
        "ec2:Describe*",
        "cloudwatch:Get*",
        "cloudwatch:Describe*",
        "cloudtrail:LookupEvents",
        "cloudtrail:DescribeTrails",
        "cloudtrail:GetTrailStatus",
        "elasticloadbalancing:Describe*",
        "logs:FilterLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

Option 2 — Broad read-only (simpler): attach AWS's managed `ReadOnlyAccess` policy instead. Simpler, but broader than necessary — and you must still add `ssm:PutParameter` (plus `kms:Encrypt`/`kms:Decrypt` if using a customer-managed key) separately, since ReadOnlyAccess doesn't include them.

Step 2 — Configure LLM, OpsGenie & SSM Parameters

Switch to user jasper with sudo su jasper

Run `configure_llm.sh`. It first creates/updates every `/jasper/config/*` SSM parameter interactively (see below), then reads the OpenRouter and OpsGenie SSM parameters and writes `model`, `api_key`, `fast_model`, `opsgenie_api_key`, and the Kubernetes/Datadog toolset-disable block into `/home/jasper/.holmes/config.yaml`.

Note: Nothing needs to be pre-created in the console. `openrouter/api_key` is mandatory and always prompts for a value. Every other parameter is optional — press N to skip. For the JSON-shaped parameters (cloudwatch, elasticsearch, freshservice, waf, sre_metrics), the script asks for each field separately and builds the JSON for you.

Reference — what each parameter looks like (you'll be prompted for these values, or their individual sub-fields, when the script runs):

Parameter Name	Type	Value (JSON where applicable)
`/jasper/config/openrouter/api_key`	SecureString	`-----`
`/jasper/config/opsgenie/api_key`	SecureString	`-----`
`/jasper/config/cloudwatch/config`	SecureString	`{"waf_log_group": "-----"}`
`/jasper/config/elasticsearch/config`	SecureString	`{"es_host": "-----", "es_user": "-----", "es_password": "-----"}`
`/jasper/config/freshservice/config`	SecureString	`{"url": "-----", "api_key": "-----"}`

Parameter Name	Type	Value (JSON where applicable)
`/jasper/config/waf/config`	SecureString	`{"webacl": "-----", "log_group": "-----"}`
`/jasper/config/sre_metrics/config`	SecureString	`{"alb_name": "-----", "target_group": "-----", "elk_endpoint": "-----", "elk_username": "-----", "elk_password": "-----"}`
`/jasper/config/teams_bot/config`	SecureString	`{"app_type": "SingleTenant", "app_id": "-----", "tenant_id": "-----", "app_password": "-----"}`
`/jasper/config/slack_bot/config`	SecureString	`{"bot_token": "-----", "app_token": "-----", "signing_secret": "-----"}`

```
sudo bash /mnt/configure_llm.sh
```

Step 3 — Teams & Slack Bot Integration

Adds a Microsoft Teams and/or Slack chat front-end to Jasper — pick one or both.

Prerequisites: for Teams, an existing Azure Bot Service app registration (App ID, Tenant ID, client secret). For Slack, a Slack App with Socket Mode enabled (Bot Token, App-Level Token, Signing Secret).

1. Run teams_slack_bot_setup.sh

Prompts for which integration(s) to set up (1: Teams, 2: Slack, 3: Both). Each syncs its codebase from S3 and starts its own systemd service (jasper-teams-bot.service / jasper-slack-bot.service). Teams optionally installs nginx + certbot to expose it over HTTPS on a domain you provide interactively (press Enter to skip if not ready). Slack uses Socket Mode — no public endpoint, nginx, certbot, or DNS record is needed at all.

```
sudo bash /mnt/teams_slack_bot_setup.sh
```

Note: Teams' optional HTTPS step assumes a DNS A record for your domain already points at this instance's public/Elastic IP, and the security group already allows inbound 443 — neither can be automated here since they require account/console access this script doesn't have.

2. Teams only — point Azure Bot Service at the new endpoint

Azure Portal → your Bot Service resource → Configuration → set Messaging endpoint to:

```
https://<your-domain>/api/messages
```

3. Teams only — package and sideload the Teams app

Generate a manifest.json + icons + zip locally (not on the server) with the included generator script, then upload it through Teams — Apps → Manage your apps → Upload a custom app:

```
python generate_teams_manifest.py \
  --bot-id <your-app-id> \
  --domain <your-domain> \
  --display-name <bot-name> \
  --short-desc "<short description>" \
  --developer "<company name>"
```

Add the app to a chat and send a message to verify — a greeting should come back immediately, followed by real answers to actual questions.

Slack only — test directly

No further setup needed. Once the service is running, find the bot in Slack and send it a direct message — Socket Mode connects automatically.

Step 4 — Test LLM Connectivity

```
holmes ask "What can you do?"
```

This should now return a real response from the model. If it doesn't, see the Troubleshooting section.

Step 5 — Sync & Register Custom Toolsets & MCP Servers

Run `setup_toolset_mcp.sh` and choose an option: 1–7 for a single custom toolset, or 8 for all 7. It downloads the selected toolset(s) from S3, registers each in `config.yaml`'s `custom_toolsets:` list, and validates every registered path actually exists on disk before finishing. Re-run it any time to add more toolsets later — it only appends what isn't already registered.

```
sudo bash /mnt/setup_toolset_mcp.sh
```

MCP Servers Follow-Up (Elasticsearch, OpsGenie, Freshservice, Multi-Account AWS)

After the toolset step above finishes (whichever of options 1–8 was chosen), the script always asks a follow-up question — press Enter to skip it entirely if this server only needs plain custom toolsets, or pick one of:

- 1: elasticsearch (MCP)
- 2: opsgenie (MCP)
- 3: freshservice (MCP)
- 4: aws-multi-account (MCP)
- 5: All MCP Servers

Adds Model Context Protocol (MCP) servers running as local systemd services, matching the production reference deployment's MCP architecture. Elasticsearch and Freshservice replace their custom_toolsets entries (MCP now serves those integrations instead); OpsGenie's MCP server runs alongside the existing native opsgenie_api_key/opsgenie_query config, not in place of it; the AWS server adds a new capability — multi-account Security Hub / Cost Explorer investigation across 4 AWS accounts via cross-account role assumption (no static credentials).

Prerequisites: configure_llm.sh has already created the elasticsearch/config, opsgenie/api_key, and freshservice/config SSM parameters — these 3 MCP servers reuse those same parameters, no new secrets are needed. mcp_servers_new/ has been uploaded once to S3 at s3://rlcatalyst-jasper-toolsets/Toolsets/mcp_servers/, the same way the custom toolsets are.

Picking a single server (1–4) fails loudly if its SSM prerequisite is missing — that's the one thing that was asked for. Picking "All MCP Servers" (5) instead skips (with a warning) whichever of elasticsearch/opsgenie/freshservice isn't configured yet, rather than installing a service doomed to crash-loop — its custom_toolsets entry, if any, is left in place in that case. Re-running this follow-up later for one specific server (e.g. after fixing its SSM parameter) only adds/updates that one entry in config.yaml's mcp_servers: block, without disturbing servers installed in an earlier run.

Note: If jasper-teams-bot.service and/or jasper-slack-bot.service are already running, this step (and the toolset step above, if it actually registered something new) automatically restarts whichever ones are active. Those bots load config.yaml once at startup and keep it in memory for their whole lifetime, so without this they'd never see toolsets or MCP servers added after they were already running.

Important: Cross-account AWS access needs a manual IAM change this script cannot make: this instance's role must be granted sts:AssumeRole on Jasper_CrossAccount_InvestigationRole in each of the 4 target accounts (aws-master, aws-9774, aws-7727, aws-9429), and each of those 4 roles' trust policy must add this instance's role ARN as a trusted principal. Both sides happen in the target accounts' IAM consoles, not on this server. Until that's done, the other 3 MCP servers (elasticsearch, opsgenie, freshservice) work normally — only cross-account AWS queries are affected.

Verify installed MCP servers are listening:

```
ss -tlnp | grep -E '18081|18082|18083|18090'
```

Step 6 — Test Each Toolset & MCP Server

Ask a natural-language question for each toolset to confirm the LLM picks the right tool.

Toolset	Sample holmes ask query
cloudtrail	List all CloudTrail trails in us-east-1
cloudwatch	What are the CPU metrics for instance i-xxxx over the last 6 hours?
elasticsearch	How many HTTP errors occurred in nginx logs in the last 24 hours?
freshservice	List open Freshservice tickets
loadbalancer	List all load balancers in us-east-1
sre_metrics	What are the current SRE metrics for prod?
waf	Show me WAF blocked request metrics for the last 2 hours

MCP Server	Sample holmes ask query
elasticsearch (MCP)	How many HTTP errors occurred in nginx logs in the last 24 hours?
opsgenie (MCP)	List open OpsGenie alerts and acknowledge the oldest one
freshservice (MCP)	List open Freshservice tickets and reply to ticket 12345
aws-api (MCP, multi-account)	List my AWS accounts, then show EC2 instances in rg-prod

Note: If a toolset was just fixed (SSM parameter created, package installed, etc.) and still fails through holmes ask, add --refresh-toolsets to force a fresh health check: holmes ask "..." --refresh-toolsets

Troubleshooting Reference

Check here first if something breaks.

"No LLM models were loaded" when running holmes ask

configure_llm.sh has not been run yet, or failed. Confirm model/api_key are present in /home/jasper/.holmes/config.yaml.

litellm.AuthenticationError / "Incorrect API key provided" referencing platform.openai.com

Do not export OPENAI_API_KEY as an environment variable anywhere. Only set model/api_key via config.yaml (handled by configure_llm.sh).

SSM parameter value fails to parse as JSON (json.decoder.JSONDecodeError)

Important: Do not wrap the JSON value in extra single quotes when pasting into the AWS Console Parameter Store value field. Store exactly: {"key": "value"} — not '{"key": "value"}'.

A toolset still fails after fixing its root cause (SSM param, package, etc.)

Jasper caches each toolset's health/prerequisite check. Force a re-check with: holmes ask "..." --refresh-toolsets

Repeated "Created fast LLM instance" lines in the output

Normal, not an error. A separate, optional feature (fast-model summarization for large tool outputs) only checks that OPENROUTER_API_KEY is present as a raw environment variable before initializing — it never reads its value, since the real key always comes from the api_key: field in config.yaml. The /usr/local/bin/holmes wrapper sets this variable to a placeholder automatically, so this line prints once per toolset that uses the feature and can be ignored. (Older installs without the placeholder instead show a "Failed to create fast LLM instance" warning, which is also harmless but was noisier — re-run install_jasper.sh to pick up the fix.)

Teams bot: AADSTS7000215 "Invalid client secret provided"

Important: The App Password stored in the teams_bot SSM parameter is the Azure AD Secret ID (a GUID) instead of the Secret Value. Delete the secret in Azure AD and create a new one; copy the Value column immediately after creation (never shown again), re-run configure_llm.sh with the correct value, then sudo systemctl restart jasper-teams-bot.service.

Teams bot: ModuleNotFoundError: No module named 'teams_integration.bot_api'

Only 2 files were synced from S3 instead of the full teams_integration/ tree. Re-upload the complete folder (bot/, bot_api/, services/, logging_config/, all __init__.py files) and re-run teams_slack_bot_setup.sh.

MCP server: AccessDenied / AssumeRole failure on a cross-account AWS query

The target account's Jasper_CrossAccount_InvestigationRole trust policy hasn't been updated to trust this instance's role ARN yet — a manual IAM console change, see the MCP Servers step. The other 3 MCP servers are unaffected.