

Red Hat AI for Intelligent Tax Administration

Red Hat, Four Inc. and Carahsoft

Chapter 1. About this guide

Artificial intelligence is changing how organizations modernize operations, improve taxpayer services and unlock the value of their data. Adopting it at scale needs a secure, flexible foundation that connects automation, hybrid cloud and intelligent workflows.

This guide is the hands-on half of that conversation. It carries eight labs that walk the whole path — standing a model up inside your boundary, teaching it your agency’s vocabulary, serving it through a filing-season peak, grounding its answers in your published guidance, automating the operational toil around it, proving the result to an authorizing official, pointing the same retrieval pattern at your own logs to drive incident diagnosis time down, and running the whole loop end to end on one platform.

Every command in these labs is real and copyable. Every lab runs in two modes: one that needs nothing but a shell, and one that executes against your own environment.

Chapter 2. The ladder: data, information, knowledge, judgement

One theme runs through the deck, the labs and this guide: a four-rung ladder from data to judgement. Each rung means the same thing everywhere it appears.

Data

What happened, uninterpreted: returns, transcripts, calls, logs, telemetry. Where it lives decides the architecture.

Information

Data in context for one case: a classification, a score, a diagnosis, a metric. What predictive AI and inference produce.

Knowledge

Information that informs an outcome: guidance, precedent, runbooks, taxonomies. The institution's memory, retrievable and cited.

Judgement

A determination someone signs. Deliberately human; automation executes what was decided, it never decides.

Every technology in this guide is tagged with the rung it serves — each lab's header table carries a Rung row, and the table below maps every product and component the labs touch. An arrow (data → information) marks a technology that converts one rung into the next; *executes judgement* marks automation, which carries out a decision a person already made. Judgement is deliberately never automated: the top rung stays with a named reviewer, an authorizing official or a governance board, and every pattern in these labs is arranged to keep it there.

Table 1. Technologies and the rung each serves

Technology	Rung
Red Hat Enterprise Linux	data
Red Hat AI Inference Server (vLLM)	data → information
Red Hat OpenShift	data
Red Hat OpenShift AI	information + knowledge
KServe / model serving	information
Granite models	information
SDG Hub	knowledge
Training Hub	knowledge
Pipelines	knowledge
Model registry	judgement
Retrieval / vector store / pgvector / Llama Stack	knowledge

Technology	Rung
Ansible Automation Platform / Event-Driven Ansible	executes judgement
Sigstore / cosign	judgement
FIPS mode	data
Compliance Operator	information
TrustyAI	information
OpenShift Logging / Loki	data
GPUs / accelerators	data
Predictive models	information
Generative models (with retrieval)	knowledge
Human reviewer	judgement
Models-as-a-Service	information
llm-d	information

Chapter 3. Who this is for

Reader	Where to start
Platform engineers	Labs 01, 03 and 05. You are the people who will operate this.
Data and application teams	Labs 02 and 04. Domain adaptation and retrieval are where mission value shows up.
Operations and SRE teams	Labs 07 and 08. The retrieval pattern from Lab 04, pointed at your own logs, runbooks and incident history — the payoff is measured in MTTR. Then the whole loop end to end: score, draft, review, run, ask.
Security, ISSO and compliance staff	Lab 06 first, then the security section of the event brief. Lab 06 is written as the checklist you would run yourself.
Programme and acquisition leads	The event brief, then the adoption path. Skip the terminals.

Chapter 4. The five outcomes, and where each is proven

Outcome	Platform capability	Lab
Modernize mission-critical operations with intelligent automation and streamlined workflows	Red Hat Ansible Automation Platform, Event-Driven Ansible, Red Hat OpenShift	Lab 05 — Automate the toil around the model , Lab 08 — One loop, end to end
Improve efficiency and accuracy by reducing manual processes and accelerating service delivery	Red Hat AI Inference Server, SDG Hub + Training Hub	Lab 01 — Serve a model inside your boundary , Lab 02 — Teach it your notice taxonomy
Unlock data-driven insights to support smarter decision-making	Red Hat OpenShift AI, vector retrieval	Lab 04 — Ground answers in your own guidance
Strengthen security and compliance with trusted, enterprise-ready AI	Red Hat Enterprise Linux FIPS mode, Compliance Operator, TrustyAI, Sigstore	Lab 06 — Prove it to your ISSO
Build an AI-ready foundation with flexible platforms designed to scale	Red Hat OpenShift AI, KServe, vLLM	Lab 03 — Survive filing season

Chapter 5. How to run the labs

Every lab is a shell script under `labs/`. The scripts run in two modes.

`LAB_MODE=simulate`

The default. Prints each command followed by the output you should expect from it. Runs on a laptop with no cluster, no GPU and no credentials. This is the mode to use during the event.

`LAB_MODE=live`

Executes each command against your own environment. The simulated output stays visible on failure, so you can see what should have happened. Requires the tools named in each lab's prerequisites.

```
# Walk a lab, one command at a time, pausing for Enter
LAB_PACE=step ./labs/lab01-rhel-ai/run.sh

# Walk every lab end to end without pausing
./bin/run-all-labs.sh

# Run for real against your own cluster
LAB_MODE=live ./labs/lab03-openshift-ai/run.sh
```

Table 2. Environment variables

Variable	Meaning	Default
<code>LAB_MODE</code>	<code>simulate</code> or <code>live</code>	<code>simulate</code>
<code>LAB_PACE</code>	<code>auto</code> or <code>step</code> (wait for Enter)	<code>auto</code>
<code>LAB_SPEED</code>	Seconds between simulated output lines	<code>0.04</code>
<code>NO_COLOR</code>	Set to any value to disable ANSI colour	<code>unset</code>



The interactive deck at slides/deck.html has a Demo view that plays all eight labs hands-free — useful when you want the room watching one screen instead of typing, and as a looping holding screen via `#demo` on the URL.



`LAB_MODE=live` runs real commands. Lab 03 includes a load test that will consume every GPU its namespace can schedule, and Lab 06 reads node configuration over SSH. Point these at a non-production cluster first.

Chapter 6. A note on the sample content

The notice families, publication references and taxonomy entries in these labs are illustrative examples written for the exercise. They are shaped like real agency content so the labs feel real, but they are not authoritative guidance and no agency has reviewed them. Replace them with your own published material before generating training data or indexing a corpus.

The numbers in the simulated output — latencies, replica counts, evaluation scores — are representative figures for a working environment, not measurements from a specific deployment. Treat them as the shape of a good result, and measure your own.

Chapter 7. Event brief

7.1. The operating reality

Tax administration has a shape that makes it unusually well suited to language models, and unusually unforgiving of the way they are normally deployed.

Demand is seasonal.

Filing season concentrates a year of volume into weeks. Capacity that sits idle in August cannot be justified; capacity that runs out in April becomes a headline.

The rules are locked in legacy code.

Notice generation, eligibility and offset logic live in systems written decades ago. Rewriting them is a decade-long programme. Wrapping them in intelligent workflows is a quarter.

The answers are in the text.

Correspondence, transcripts, published guidance and case notes are unstructured. Traditional analytics never reached them. Language models are built for exactly this material.

The data cannot leave.

Federal tax information carries handling rules that rule out sending prompts to a public endpoint. The model has to come to the data — on premises, in an accredited cloud region, or in an air-gapped enclave.

That last constraint is the one that decides the architecture. Choosing a model is a six-month decision. Choosing where models run, how they are governed and how they scale is a ten-year one.

7.2. Where AI lands in the filing lifecycle

Stage	Work	Good first candidate?
Intake and classification	Route paper, portal and phone contacts by intent rather than keyword	Yes
Validation	Flag inconsistencies and missing documentation before a case is opened	Yes
Correspondence	Draft plain-language notices for a reviewer to edit and sign	Yes
Exam selection	Determinations affecting a specific taxpayer	No — governance first, automation much later
Collections support	Summarize case history so an officer opens a case already briefed	Yes
Taxpayer service	Answer from published guidance, with the citation attached	Yes

Two rules of thumb come out of that table.

1. **Start where a human already reviews the output.** If a person signs the letter today, a model that drafts it changes throughput without changing accountability. That is a defensible first deployment.
2. **Automate the toil around the model, not just the model.** Most of the effort in production AI is provisioning, patching, scaling and evidence collection. That is an automation problem, and it is already solved.

7.3. Reference architecture

The platform is four horizontal layers with two concerns running vertically through all of them. Each layer is noted with the rung of the ladder — data, information, knowledge, judgement — that it serves.

Table 3. Layers, bottom to top

Layer	What it provides
Red Hat Enterprise Linux and the Red Hat AI Inference Server	Bare metal GPU nodes, virtualized datacenter, accredited public cloud or a classified enclave. Granite models, accelerator drivers and the vLLM-based inference server ship as one supported stack. (rung: data)
Red Hat OpenShift	Scheduling, GPU partitioning, network policy, multi-site placement and GitOps delivery. The same cluster runs the modernized case-management applications beside the models they call. (rung: data)
Red Hat OpenShift AI	Model serving on KServe and vLLM, tuning and evaluation pipelines, a model registry with versions and lineage, and retrieval against a vector store. (rung: information + knowledge)
Mission applications	Correspondence intake, notice drafting, case summarization, guidance assistant. Systems of record stay where they are, reached through APIs. (rung: judgement)

Concerns, running through every layer

Automation

Red Hat Ansible Automation Platform and Event-Driven Ansible provision, patch, remediate and collect evidence. Covered in [Lab 05 — Automate the toil around the model](#). On the ladder, Automation executes judgement: it carries out what a person decided, and it never decides.

Trust

Signed models, FIPS-validated cryptography, SELinux, the Compliance Operator, TrustyAI drift and fairness monitoring, and disconnected mirroring. Covered in [Lab 06 — Prove it to your ISSO](#). On the ladder, Trust produces the evidence (information) an authorizing official signs against: signatures, scan results and drift figures are what make the judgement defensible.

The point of drawing the two concerns vertically is that neither is a layer you add at the end. A project that treats compliance as a phase after deployment discovers, at the worst possible moment, that it cannot produce evidence for anything that already happened.

7.4. The questions an authorizing official will ask

Question	Platform control	Where
Where does our data go when someone prompts the model?	Nowhere. Inference runs on your cluster, in your enclave, on your accelerators.	Lab 01 — Serve a model inside your boundary
Can you prove this model is the one we approved?	Model artifacts are signed and verified before admission to the cluster.	Lab 06 — Prove it to your ISSO
Is the cryptography validated?	FIPS mode is a supported operating state of the OS and the platform above it.	Lab 06 — Prove it to your ISSO
How do we know it stays compliant next quarter?	Scheduled scans against a hardening profile, with machine-readable results.	Lab 06 — Prove it to your ISSO
How do we detect the model drifting or skewing?	Continuous drift and fairness metrics on live inference traffic.	Lab 06 — Prove it to your ISSO
What if we have no internet at all?	Mirror images and models into the enclave; the platform is built for disconnected operation.	Lab 01 — Serve a model inside your boundary
Who is accountable when it is wrong?	A named reviewer. The deployment patterns here keep a person on the signature.	Lab 04 — Ground answers in your own guidance

7.5. Adoption path

Prove — days 0 to 90

- Stand up one supported endpoint with the Red Hat AI Inference Server
- Agree the accelerator capacity plan and where it physically sits
- Settle which data classes may be used, in writing
- Pick one text-heavy workflow that already has a human reviewer
- Measure the baseline before changing anything

Ground — days 90 to 180

- Move serving onto Red Hat OpenShift AI with autoscaling
- Add retrieval over your own published guidance
- Tune on agency vocabulary with SDG Hub and Training Hub
- Wire drift, fairness and compliance monitoring

- Automate provisioning and patching end to end

Scale — days 180 and beyond

- Promote through a model registry with real approvals
- Add workflows two through five on the same platform
- Prove the filing-season surge with a load rehearsal
- Extend to the disconnected or classified enclave
- Report outcomes in mission terms, not token counts



Most programmes stall on the second and third bullets of the Prove phase — no accelerator capacity plan, and no written agreement on which data may be used. Neither is a technical problem, and both take longer than the pilot.

Chapter 8. AI fundamentals — the theory, demystified

Everything in the labs works better when the room shares a mental model of what a language model actually does. This chapter is that mental model, in plain language. Nothing in it requires mathematics beyond multiplication.

8.1. What actually happens when you ask

Take the question a facilitator asks in Lab 01: *"Find the missing document for this contact."* Four things happen, and only four.

1. **Your words become tokens.** The model never sees words. Text is split into small pieces called tokens — roughly four characters each — and every token becomes a number. "Form 8822" is three tokens; a page of a notice is about 700.
2. **The model scores what could come next.** One pass through billions of learned numbers — the *weights* — turns your tokens into a probability for every possible next token.
3. **It writes one token at a time.** The winning token is appended to the input and the whole pass runs again, until the answer is complete. That is all "generation" is.
4. **Temperature is the dial.** At 0.1 the model almost always takes the top candidate: predictable and repeatable, which is what tax administration wants.

The demystifying sentence, worth saying out loud in the session: when Lab 01 returns `missing_doc: Form 8822`, that answer was not fetched from any system. It is the most probable continuation of the contact text under those weights — nothing else happened, and nothing else needed to.

8.2. Where the answer comes from

The question underneath most audience confusion is: *"do I need to do anything to get that response?"* The answer is no — sending the prompt is the request. There are exactly three places an answer can draw from:

Source	What it contributes
The weights	Patterns learned in training, then frozen. Not a database — nothing is looked up, nothing is fetched from anywhere. Lab 02 is the only lab that changes them.
The prompt	Everything you send: the taxpayer's message plus your instructions. The system prompt in Lab 01's <code>inquiry.json</code> is what names the fields to fill — read it and the output format stops being mysterious.
Retrieval (optional)	Your own documents, found by meaning and placed into the prompt. This is how answers arrive with citations — or refuse honestly. Lab 04 builds it.

The prompt and any retrieved passages share the *context window* — the model's short-term

memory, cleared after every request.

Three reassurances follow directly from this mechanism, and they are worth stating explicitly to any audience handling taxpayer data:

- **Nothing leaves.** No internet call happens at answer time. On this platform the model runs on your hardware — Lab 01 proves it with one line of `ss` output.
- **Nothing is learned from you.** A prompt does not train the model. Weights change only in deliberate tuning, with data you choose and review.
- **Confident nonsense is possible.** Plausible tokens can assemble a wrong answer. That is why retrieval cites, thresholds refuse, and a human signs.

8.3. How this maps onto the labs

Lab	The theory it demonstrates
01	Pure inference: tokens in, predicted tokens out, steered by a system prompt. The <code>tokens: 214 in / 96 out</code> line is the whole mechanism made visible.
02	The only weight change in the series: a LoRA adapter tuned with Training Hub on synthetic data SDG Hub expanded from reviewed seed examples. Explains why the base model scored 0.61 on notice accuracy and the tuned one 0.94 — CP2000 routing was never in the base training data.
03	Inference at scale: GPUs matter because each pass is billions of multiplications; vLLM batches many requests through each pass; the 90-second cold start is loading 8 GB of weights into GPU memory.
04	Embeddings place similar meanings near each other in a vector space; retrieval finds the nearest guidance passages and puts them in the context window. Below the similarity threshold, the honest output is a refusal.
05	No machine learning at all — deterministic automation around the model. The intelligence stays in the model; the reliability lives in the automation.
06	Weights are frozen, so a deployed model cannot decay — drift metrics detect the world changing, not the model. And a model is ultimately a file: hash it, sign it, verify it.
07	No new machine learning at all: Lab 04's retrieval pattern pointed at a different corpus. Logs, runbooks and postmortems become vectors; the question finds the nearest evidence; the model assembles a cited diagnosis. The hours lived in diagnosis — grounded retrieval removes them.

8.4. Glossary

Term	Plain-language meaning
Token	The small piece of text a model actually reads and writes — roughly four characters. "Form 8822" is three tokens.

Term	Plain-language meaning
Tokenization	Splitting text into tokens and turning each into a number — the only translation step between words and mathematics.
Weights (parameters)	The billions of numbers learned during training. Patterns of language, not a database of facts; frozen once deployed.
Inference	Running the model: tokens in, one predicted token out, repeated.
Prompt	Everything you send — instructions plus the text to work on. Sending it is the whole request.
Context window	How much the model can consider at once. Short-term memory, cleared after every request.
Temperature	How adventurous the next-token choice is. Near zero: repeatable — the right setting for tax work.
Embedding	Text converted into a vector, where similar meanings sit close together. What makes searching by meaning possible.
Vector store	A database of embeddings — Labs 04 and 07 run one in pgvector.
RAG	Retrieval-augmented generation: fetch relevant passages, place them into the prompt, require the answer to cite them.
Fine-tuning (LoRA)	Deliberately adjusting weights with your examples. LoRA trains a small add-on instead of all weights — cheaper, reversible.
Hallucination	A confident answer assembled from plausible tokens with nothing behind it. Countered by retrieval, thresholds, refusal, and a human signature.
Drift	The input mix changing versus the accredited baseline. The world changed — not the model. Weights cannot rot.



The interactive deck carries this material as its four foundations slides ("AI in plain English", "Inside an LLM", "Data to judgement", "Two kinds of AI"), a theory panel inside every lab, and this glossary at the end of the Labs view — in English and Spanish, with a light and a dark theme.

Chapter 9. Lab 01 — Serve a model inside your boundary

Product	Red Hat AI Inference Server
Rung	data → information
Time	About 5 minutes
Script	labs/lab01-rhel-ai/run.sh
Outcome	A supported LLM classifying taxpayer contacts on hardware you control

9.1. Objective

Bring up a large language model on a single GPU host and use it to classify a real taxpayer inquiry — with no inference call leaving the machine.

This lab exists to answer one question directly, because it is the question that decides whether anything else in this guide is relevant to a tax agency: **where does our data go when someone prompts the model?**

The serving engine is the Red Hat AI Inference Server: the vLLM engine, packaged and supported as a signed container that runs anywhere Red Hat Enterprise Linux runs. (Earlier versions of this lab used the retired InstructLab `ilab` CLI; the endpoint it exposes — an OpenAI-compatible API on localhost — is unchanged.)

On the ladder this lab is the data → information step: an uninterpreted taxpayer contact goes in, and inference on hardware you control turns it into a classification, a routing queue and a draft for one case — the data never moves, which is exactly why where it lives decides the architecture.

9.2. Prerequisites

For `LAB_MODE=simulate`, nothing. For `LAB_MODE=live`:

- A Red Hat Enterprise Linux host with a supported accelerator and the container toolkit's CDI devices generated (`nvidia-ctk cdi generate`)
- `podman` and `huggingface-cli` on the path
- Registry credentials for `registry.redhat.io`
- Roughly 20 GB of free disk for the model

```
cd labs/lab01-rhel-ai
LAB_PACE=step ./run.sh
```

9.3. Walkthrough

Step	Command	What to look for
1	<code>podman pull registry.redhat.io/rhais/...</code>	The signature check on the way in. The serving engine is signed content, patched and lifecycled like any other RHEL container.
2	<code>huggingface-cli download RedHatAI/...</code>	A validated model from the Red Hat AI repository, landing on local disk. For an enclave, mirror it into your own registry instead — Lab 06 verifies the signature.
3	<code>podman run ... -p 127.0.0.1:8000:8000 ...</code>	The vLLM engine starting and the port publication. In production this is a Quadlet-managed systemd unit, not a foreground process.
4	<code>ss -ltnp grep :8000</code>	The point of the lab. The endpoint is published on <code>127.0.0.1</code> . There is no egress path from this process to a public model API.
5	<code>curl .../v1/chat/completions</code>	A structured classification with a routing queue, the missing form, and a draft reply written for a human to edit and sign.

9.4. What good looks like

The model returns a category, a routing queue, the missing document and a draft reply. It does **not** state a determination as final — that behaviour comes from the system prompt in `assets/inquiry.json`, which is worth reading before you change it.

Latency on a single H100 for an 8B model at this prompt length lands under two seconds. If yours is much slower, check that the model actually loaded onto the accelerator rather than falling back to CPU.

9.5. Troubleshooting

Error: setting up CDI devices: unresolvable CDI devices `nvidia.com/gpu`

The container toolkit has not generated CDI specs on this host. Run `nvidia-ctlk cdi generate --output=/etc/cdi/nvidia.yaml` and retry.

Pull fails with an authentication error

Run `podman login registry.redhat.io` first. The lab does not handle credentials, deliberately.

The model loads but inference is slow

Check `nvidia-smi` during a request. If GPU utilization stays near zero, the runtime fell back to CPU — usually a driver or container toolkit mismatch.

Port 8000 already in use

Another serving process is running. `ss -ltnp` will name it.

9.6. Discussion questions

1. Which host in your estate would run this, and who owns its accreditation?
2. What would have to be true for this endpoint to be reachable from your case-management application — and what network policy would you want around it?
3. The base model here knows nothing about your notice codes or routing queues. Who in your organization owns that knowledge today, and in what format?

That last question is [Lab 02 — Teach it your notice taxonomy](#).

Chapter 10. Lab 02 — Teach it your notice taxonomy

Product	SDG Hub + Training Hub on Red Hat AI
Rung	knowledge
Time	About 6 minutes to walk; hours of GPU time to run for real
Script	labs/lab02-customization/run.sh
Outcome	A tuned model that knows your notice families, queues and house style

10.1. Objective

The base model from [Lab 01 — Serve a model inside your boundary](#) does not know your agency's notice codes, your routing queues or how your correspondence is meant to read. Add that knowledge as seed YAML a subject matter expert can write and a reviewer can approve, generate synthetic training data from it with the SDG Hub library, fine-tune with Training Hub, and prove the result is better than what you started with.

On the ladder this lab produces knowledge: the seed YAML is guidance the institution owns — notice families, routing rules and house style, written by the people who apply them and reviewed like code — and tuning carries that institutional memory into the model so it informs every outcome.

10.2. Why this matters more than it looks

Domain adaptation is usually where AI programmes stall. The conventional path needs a labelled dataset, which needs annotators, which needs a budget and six months. The seed-example approach inverts it: a handful of examples written by the people who already own the knowledge become thousands of training samples, and the contribution is reviewed like code. (This workflow began life in InstructLab; the retired `ilab` CLI is gone, and its ideas carry forward as the composable `sdg_hub` and `training_hub` Python libraries.)

The file at [labs/lab02-customization/taxonomy/knowledge/gov/tax/notices/qna.yaml](#) is worth reading in full. It is deliberately written the way a real contribution would be — plain language, one notice family per context block, and a house-style section that captures the rules a correspondence reviewer applies by instinct.

10.3. Prerequisites

For live mode: a GPU host, Python with the `sdg-hub` and `training-hub` libraries, the [Lab 01 — Serve a model inside your boundary](#) endpoint running as the generating model (the "teacher" in SDG Hub's terms — the model that writes the new examples), and several hours of GPU time for the training step.

```
cd labs/lab02-customization
LAB_PACE=step ./run.sh
```

10.4. Walkthrough

Step	Command	What to look for
1	<code>cat .../qna.yaml</code>	Knowledge written as prose, not as feature vectors. A tax specialist can review this. That is the whole design goal.
2	<code>python3 check_seeds.py</code>	Schema validation. Put this step in CI. A malformed contribution should fail in a pull request, not after four hours of GPU time.
3	<code>python3 generate.py --endpoint http://127.0.0.1:8000/v1</code>	12 seed examples becoming roughly 2,200 kept samples after quality filtering, generated against the Lab 01 — Serve a model inside your boundary endpoint. Every sample traces back to something a human wrote.
4	<code>python3 train.py --method lora</code>	Loss dropping across three epochs. The checkpoint path is what you promote.
5	<code>python3 evaluate.py</code>	The comparison table against the base model. This is the artifact your governance board actually wants.

10.5. What good looks like

The evaluation table shows meaningful gains on agency-specific tasks — notice family accuracy, correct routing queue, citing the right form — and a smaller gain on general readability. That pattern is the healthy one. If general benchmarks move a lot, you have probably overfitted to the seed examples.

Table 4. Representative evaluation output

Measure	Base	Tuned	Delta
Notice family accuracy	0.61	0.94	+0.33
Correct routing queue	0.58	0.91	+0.33
Cites the right form	0.44	0.89	+0.45
Plain-language readability	0.72	0.86	+0.14
Overall judge score	6.1	7.8	+1.7



Keep the held-out evaluation set **out** of the seed data. If the questions you evaluate on are in the material you generated training data from, the numbers are meaningless and someone will eventually notice.

10.6. Troubleshooting

`check_seeds.py` reports a schema error

Check `version: 3` at the top and the indentation under `questions_and_answers`. Nearly every failure is one of those two.

Data generation produces very few samples

The quality filter is discarding them. Usually the seed contexts are too short or too similar to each other. Aim for a paragraph of real context per example. Also confirm the [Lab 01 — Serve a model inside your boundary](#) endpoint is up — generation quality is bounded by the model doing the generating.

Training runs out of memory

Lower the effective batch size, or use the LoRA method rather than full fine-tuning. The lab uses LoRA rank 32.

The tuned model regresses on general questions

Too many epochs, or too narrow a seed set. Add skills examples alongside knowledge examples.

10.7. Discussion questions

1. Who at your agency writes the equivalent of this YAML, and would they be willing to review a pull request?
2. What is your held-out evaluation set, and who decides when a score is good enough to promote?
3. What would you need to see before letting a tuned model draft correspondence that a reviewer signs?

Chapter 11. Lab 03 — Survive filing season

Product	Red Hat OpenShift AI with KServe and vLLM
Rung	information at scale
Time	About 6 minutes
Script	labs/lab03-openshift-ai/run.sh
Outcome	An autoscaling inference service that absorbs an April load rehearsal

11.1. Objective

One GPU host is a pilot. Filing season is a capacity event. Deploy the tuned model from [Lab 02 — Teach it your notice taxonomy](#) with KServe and vLLM, then run a load rehearsal and watch the platform scale from two replicas to twenty and back down again — with no application change.

On the ladder this lab serves information at scale: every request is one classification for one case, and the platform’s job is to keep producing that information at filing-season volume without changing what the data is, where it lives or who signs the result.

11.2. The manifest is the capacity plan

The single most important file in this lab is [labs/lab03-openshift-ai/manifests/inferenceservice-tax-classifier.yaml](#). Read it before you run anything. Four decisions in it are worth arguing about with your own team:

minReplicas: 2

Not zero. Cold-starting an 8B model costs roughly 90 seconds, which is not a number you want on a taxpayer-facing path. Scale-to-zero is for batch work.

scaleMetric: concurrency with scaleTarget: 24

Scaling on concurrent requests rather than CPU. GPU inference saturates on in-flight requests long before CPU looks busy, so CPU-based autoscaling reacts far too late.

scale-down-delay: 5m

Holds capacity for five minutes after a spike so a bursty afternoon does not thrash the scheduler.

storageUri pinned to a digest

[@sha256:...](#), not a tag. The digest is what the authorizing official approved. A mutable tag means the thing you accredited and the thing you are serving can silently diverge — which is exactly the failure [Lab 06 — Prove it to your ISSO](#) checks for.

11.3. Prerequisites

For live mode: an OpenShift cluster with Red Hat OpenShift AI installed, GPU nodes with the NVIDIA GPU Operator, [oc](#) authenticated, and [hey](#) for the load test.

```
cd labs/lab03-openshift-ai
LAB_PACE=step ./run.sh
```

11.4. Walkthrough

Step	Command	What to look for
1	<code>oc new-project</code>	A dedicated namespace. Quota and network policy attach here.
2	<code>oc get servingruntimes</code>	vLLM is already there. You are not building a serving stack.
3	<code>oc apply -f manifests/...</code>	The deployment is a reviewable file in git.
4	<code>oc get inferencservice -w</code>	Ready at roughly 90 seconds. That number is your cold-start cost.
5	<code>curl .../v1/completions</code>	One classification, sub-second.
6	<code>hey -z 90s -c 400</code>	Replicas climbing 2 → 20, latency holding, zero errors, then scaling back to the floor.

11.5. What good looks like

Under 400 concurrent requests the service holds p95 latency under a second, returns no errors, and reaches its replica ceiling within about 40 seconds. Afterwards it returns to `minReplicas` once the scale-down delay expires.

The behaviour to watch for is the **shape**, not the absolute numbers: fast scale-up, stable latency through the ramp, and a return to floor that does not happen so fast it re-triggers.



This load test will consume every GPU the namespace is allowed to schedule, for 90 seconds. Run it against a non-production cluster, or during a window your platform team has agreed to.

11.6. Troubleshooting

InferenceService never becomes Ready

`oc describe inferencservice tax-classifier`. The usual causes are no schedulable GPU node, a missing image pull secret for the model registry, or a digest that does not exist.

Ready, but requests time out

Check `oc logs` on the predictor pod for the vLLM startup line. A model that exceeds GPU memory will start and then stall.

Replicas do not scale up under load

Confirm the KServe autoscaler is reading concurrency, and that the namespace quota permits more GPUs. Hitting a quota ceiling looks exactly like a broken autoscaler.

Scaling oscillates

Increase **scale-down-delay**. Thrashing usually means capacity is being released faster than the arrival pattern warrants.

11.7. Discussion questions

1. What is your actual filing-season concurrency, and how would you measure it before you need it?
2. Where do the GPUs physically live, and does that placement satisfy your data handling rules?
3. Who approves a change to this manifest, and is that the same person who approves a change to the model?

Chapter 12. Lab 04 — Ground answers in your own guidance

Product	Red Hat OpenShift AI with a pgvector store
Rung	knowledge
Time	About 5 minutes
Script	labs/lab04-rag/run.sh
Outcome	Answers that carry their citation — and a refusal when they cannot

12.1. Objective

A model that sounds confident and cites nothing is a liability in a tax agency. Put your published guidance behind the model with retrieval, so every answer returns the passage it came from — and so a question the corpus cannot answer produces a referral rather than a guess.

On the ladder this lab is knowledge: published guidance is the institution's memory, and retrieval makes it retrievable and cited, so every answer informs an outcome with the passage it came from — or refuses and refers to a person when the memory does not cover the question.

12.2. The refusal is the point

Most retrieval demos show the good answer. This lab spends half its time on the bad question, because that is the behaviour that decides whether a governance board will let this near a taxpayer.

[labs/lab04-rag/query.py](#) carries a `RELEVANCE_THRESHOLD` of 0.75. Below it the script refuses, recommends referral to a specialist, and exits with code 3 so a calling workflow can route the contact to a human. A retrieval system that always answers is worse than no retrieval system, because it launders a guess into something that looks sourced.

Step 5 prints the retrieval trace, showing every rejected candidate and its score. Show that screen to your assessors. It turns "the model refused" into "here is exactly why, and here is the number you can change."

12.3. Prerequisites

For live mode: an OpenShift cluster, `psycopg` installed, an embeddings endpoint (the one from [Lab 01 — Serve a model inside your boundary](#) or [Lab 03 — Survive filing season](#)), and a corpus of your own guidance in [labs/lab04-rag/corpus/](#) as `.txt`, `.md`, `.adoc` or `.html`.

```
cd labs/lab04-rag
LAB_PACE=step ./run.sh

# Chunk your own corpus without touching a database
```

```
python3 ingest.py --source ./corpus --chunk 800 --overlap 120 --dry-run
```

12.4. Walkthrough

Step	Command	What to look for
1	<code>oc apply -f manifests/pgvector.yaml</code>	The vector store runs on the same cluster as the model. Nothing leaves it.
2	<code>python3 ingest.py</code>	Chunk counts per publication. Re-run this when guidance is republished.
3	<code>python3 query.py '...CP14...'</code>	A correct answer and a <code>sources:</code> block with scores.
4	<code>python3 query.py '...2031...'</code>	A refusal, and a recommendation to refer. Exit code 3.
5	<code>python3 query.py --explain</code>	The retrieval trace: every candidate, its score, and the threshold.

12.5. Design notes worth arguing about

Chunking is deliberately boring

Fixed window with overlap, preferring a paragraph break near the edge. Published guidance is already structured into short numbered sections, and clever splitters tend to break that structure rather than respect it.

Retrieval keeps sensitive text out of training

The corpus is queried, never learned. That distinction matters enormously to a data governance review, and it is worth stating explicitly in writing.

Guidance changes more often than weights should

Re-ingesting a republished publication is minutes. Retraining is not. When the answer needs to change because the rule changed, retrieval is the right layer.

12.6. Troubleshooting

`psycopg` is not installed

`pip install 'psycopg[binary]'`, or run the lab in simulate mode.

Embeddings endpoint unreachable

The scripts default to <http://127.0.0.1:8000/v1/embeddings>. Override with `EMBED_ENDPOINT`, and confirm the model from [Lab 01 — Serve a model inside your boundary](#) is still serving.

Everything scores below the threshold

Either the corpus does not cover the question, or the embedding model differs between ingest and query. Those must match — a mismatch produces uniformly low scores that look like an empty corpus.

Answers cite the right passage but read poorly

That is a prompt problem, not a retrieval problem. The system prompt is at the top of `query.py`.

12.7. Discussion questions

1. Which of your publications would you index first, and who owns their currency?
2. What should happen operationally when the model refuses — which queue does that contact land in?
3. Where would you set the threshold, and who gets to change it?

Chapter 13. Lab 05 — Automate the toil around the model

Product	Red Hat Ansible Automation Platform with Event-Driven Ansible
Rung	executes judgement
Time	About 5 minutes
Script	labs/lab05-ansible/run.sh
Outcome	Saturation detected and remediated in about a minute, with an audit trail

13.1. Objective

Most of the effort in production AI is not the model. It is provisioning, patching, scaling and evidence collection. Wire a health check that tests the endpoint the way a taxpayer would, then an Event-Driven Ansible rulebook that remediates saturation before anyone opens a ticket.

On the ladder this lab executes judgement: the rulebook carries out decisions the platform team already made — which conditions to remediate and which to escalate — and it never makes a new one, which is exactly why the provenance rule pages a person instead of redeploying.

13.2. The rulebook is the runbook

[labs/lab05-ansible/rulebooks/inference_saturation.yml](#) maps four conditions to four job templates. There is no separate runbook document to go stale, because this file **is** the runbook and it executes.

One rule in it is deliberately different from the others. When the served model digest stops matching the approved change record, the rulebook escalates rather than self-remediating:

```
- name: Served model digest no longer matches the change record
  condition: event.alert.labels.alertname == "ModelProvenanceMismatch"
  action:
    run_job_template:
      # Deliberately does NOT self-remediate. A provenance mismatch is a
      # security event: page a human, do not quietly redeploy over it.
      name: "Escalate provenance mismatch"
```

Automatically redeploying over a provenance mismatch would destroy the evidence of what was actually running. That is a decision worth making explicitly, and worth defending in a design review.

13.3. The digest check people forget

The health check playbook does four things, and the fourth is the one that usually gets left out: it asserts that the model actually being served matches the digest in the approved change record. A

model that has drifted from the approved artifact is an accreditation finding, not an incident, and it will be found by an assessor if it is not found by automation first.

13.4. Prerequisites

For live mode: Red Hat Ansible Automation Platform 2.4 or later, `ansible-navigator` and `ansible-rulebook` on the path, the `ansible.eda` collection, an Automation Controller with the named job templates, and Alertmanager configured to post to the rulebook source.

```
cd labs/lab05-ansible
LAB_PACE=step ./run.sh
```

13.5. Walkthrough

Step	Command	What to look for
1	<code>ansible-navigator run playbooks/model_endpoint_health.yml</code>	Four tasks: the model list, a canary inquiry, the latency assertion, and the digest match against <code>CR-8841</code> .
2	<code>cat rulebooks/inference_saturation.yml</code>	Four conditions, four job templates, one of which deliberately escalates instead of fixing.
3	<code>ansible-rulebook --rulebook ...</code>	An alert at 14:38:02, remediation launched at 14:38:03, resolved at 14:39:04. Sixty-two seconds, no pager.
4	<code>awx job list</code>	The audit trail, with a launching identity on every row — including <code>eda-rulebook</code> for the automated action.

13.6. What good looks like

The alert-to-resolved interval is under two minutes and no human touched it. More importantly, the `awx job list` output at the end is a complete record of what changed, when, and what launched it — assembled continuously rather than reconstructed at the end of a quarter.

That job list is the evidence [Lab 06 — Prove it to your ISSO](#) depends on. Automation and compliance are the same investment described from two directions.

13.7. Troubleshooting

`ansible-rulebook: command not found`

Install `ansible-rulebook` and a JDK — the rules engine needs Java. Or run in simulate mode.

Rulebook starts but no events arrive

Confirm Alertmanager is posting to the rulebook host and port, and that nothing between them

is dropping the webhook.

Job template not found

The names in the rulebook must match the job templates in Controller exactly, including the organization.

The canary assertion fails intermittently

`canary_threshold_seconds` is 2.0 in the playbook. If the endpoint is genuinely that slow under load, that is a capacity finding — raising the threshold hides it rather than fixing it.

13.8. Discussion questions

1. What do you check today after a model deployment, and how much of it is manual?
2. Which conditions at your agency should escalate rather than self-remediate?
3. Who would you let launch an automated capacity change in production, and what approval sits in front of it?

Chapter 14. Lab 06 — Prove it to your ISSO

Product	Red Hat Enterprise Linux, Compliance Operator, TrustyAI, Sigstore
Rung	evidence → judgement
Time	About 6 minutes
Script	labs/lab06-trust/run.sh
Outcome	The four checks an authorizing official actually runs, with real results

14.1. Objective

Walk the exact checks an authorizing official will run against an AI workload: is this the model we approved, is the cryptography validated, is the platform still hardened this quarter, and is the model behaving the way it did at accreditation.

Every one of these is a platform feature you switch on. None of them is a project you fund.

On the ladder this lab runs evidence → judgement: every check produces information — a signature result, a FIPS state, a scan finding, a drift figure for this workload — and the authorizing official's decision to accept that evidence is judgement, a determination a person signs and no platform feature makes for them.

14.2. Prerequisites

For live mode: `oc` authenticated to a cluster with the Compliance Operator and TrustyAI installed, `cosign` on the path, the agency signing public key, and SSH access to a GPU node.

```
cd labs/lab06-trust
LAB_PACE=step ./run.sh
```

14.3. Walkthrough

Step	Check	What to look for
1	<code>cosign verify</code>	The signature validates and names the signer, the timestamp and the change record. Verification happens before admission , not as an audit afterwards.
2	<code>fips-mode-setup --check</code> and <code>sestatus</code>	FIPS enabled, SELinux enforcing. Both are supported operating states, not hardening you bolt on.
3	<code>oc get compliancescan</code>	A scan result — including NON-COMPLIANT , which is the honest outcome.

Step	Check	What to look for
4	<code>oc get compliancecheckresult ... FAIL</code>	Exactly which control failed, with a remediation available.
5	<code>TrustyAI metrics</code>	Drift and fairness figures on live inference traffic, compared against the accreditation baseline.

14.4. Why the failing check is in the lab on purpose

Step 3 returns **NON-COMPLIANT** for the `ocp4-moderate` profile, and step 4 shows that exactly one of 217 checks failed: audit log forwarding.

A demo where every scan passes teaches the wrong lesson. Real clusters have findings, and the useful skill is reading a result and deciding what to do about it. There are two ways to fix this one — apply the remediation directly, or let the [Lab 05 — Automate the toil around the model](#) rulebook apply it, which produces the same fix plus a job record with a launching identity. The second is the one your assessor wants.

14.5. Reading the drift numbers correctly

The TrustyAI output shows `channel_mix` drift at 0.212 against a 0.150 threshold. The instinct is to conclude the model is failing. It is not.

Input distribution shifted because more taxpayers moved to the portal. Nothing is wrong with the model at all — drift monitoring is telling you that the **world** changed, which is exactly what you want it to tell you. The response is to investigate the input, decide whether the accreditation baseline is still representative, and re-baseline deliberately if it is not.

Confusing "the inputs changed" with "the model broke" is the most common misreading of these metrics, and it leads teams to retrain when they should have been asking a business question.

14.6. The four alerts worth wiring

`labs/lab06-trust/manifests/trustyai-service.yaml` defines three Prometheus rules alongside the TrustyAI service:

ModelDriftDetected

Input distribution has moved from the baseline. Warning, after 30 minutes — long enough not to fire on a lunchtime spike.

FairnessOutOfBand

Disparate impact ratio outside the agreed band. Critical. The annotation says to route affected contacts to human review until resolved, which is an operational decision that belongs in the alert, not in someone's memory.

ModelProvenanceMismatch

The served digest no longer matches the approved change record. Critical, and handled by the [Lab 05 — Automate the toil around the model](#) rulebook, which escalates rather than self-remediating.

14.7. Troubleshooting

cosign verify fails on a model you believe is correct

Check you are verifying by digest rather than tag, and that the public key matches the signing key used at release.

No compliancescan resources exist

The Compliance Operator is installed but no `ScanSettingBinding` has been created. Nothing runs until you bind a profile.

TrustyAI reports no metrics

It needs inference traffic and a configured baseline. A freshly installed service with no traffic reports nothing, which looks like a broken install.

FIPS check says disabled

FIPS mode is set at installation. It is not a runtime toggle, and retrofitting it is a rebuild.

14.8. Discussion questions

1. Who at your agency signs model artifacts today, and where does that key live?
2. Which hardening profile applies to this workload, and who accepted the current set of findings?
3. What is your accreditation baseline for model behaviour, and what process re-baselines it when the world legitimately changes?

Chapter 15. Lab 07 — Ask your own logs

Product	Red Hat OpenShift AI, OpenShift Logging (Loki), Granite
Rung	information → knowledge
Time	About 5 minutes
Script	labs/lab07-ops-notebook/run.sh
Outcome	An on-cluster notebook over your own operational data, and the MTTR case for it

15.1. Objective

Aim the [Lab 04 — Ground answers in your own guidance](#) retrieval pattern at operations itself. Cluster logs, runbooks, incident history and change records become a private notebook you can question during a live incident — a NotebookLM for the platform, running on the platform. Then read the numbers: the payoff is measured in mean time to restore, because diagnosis is where the hours live.

On the ladder this lab climbs from information to knowledge: the log line is data, the cited diagnosis is information about one incident, and the runbooks, postmortems and change records it draws on are the institution's memory — a generated summary joins that memory only after a person reviews it.

15.2. Why this matters more than it looks

When something breaks at 2 a.m., the answer usually already exists — in a runbook, a past incident's postmortem, a change record from that afternoon, or a log line from the failing pod. What costs hours is that those four things live in four systems and three people's heads. A notebook grounded in all of them answers the on-call question — **what changed, has this happened before, which runbook applies** — in seconds, with citations the engineer can check before acting.

Two properties make this defensible in a tax agency:

- **It is the same stack as the taxpayer-facing system.** Nothing new is installed; only the corpus changes. One retrieval pattern, two corpora, one governance story.
- **It stays inside the boundary.** Logs carry hostnames, account fragments and case identifiers — FTI-adjacent by nature. The notebook runs on the same OpenShift install it observes; nothing is exported to a hosted service.

15.3. Prerequisites

For live mode: an OpenShift cluster with OpenShift Logging (Loki), the [Lab 04 — Ground answers in your own guidance](#) vector store and model serving in place, `logcli`, and Python for the ingest and reporting helpers.

```
cd labs/lab07-ops-notebook
LAB_PACE=step ./run.sh
```

15.4. Walkthrough

Step	Command	What to look for
1	<code>logcli query '{namespace="tax-inference"} = "ERROR"'</code>	The incident: OOM errors and restart loops on the predictor pods. This is the raw material the notebook will read.
2	<code>python3 ingest_ops.py</code>	Four corpora going into one index: Loki export, runbooks, postmortems, change records. Same vector store as Lab 04 — Ground answers in your own guidance — only the corpus changed.
3	<code>llama-stack-client inference chat-completion ...</code>	A cited diagnosis: the change record that caused it, the past incident that matches it, and the runbook section that fixes it. Verify, then act.
4	<code>awx job_templates launch ...</code>	Remediation through the Lab 05 — Automate the toil around the model automation, so the fix lands in the controller's job history — that list is your evidence trail.
5	<code>python3 mttr_report.py</code>	The decomposition. Detect, fix and verify are unchanged; diagnosis falls from hours to minutes. That is the whole argument.

15.5. What good looks like

The diagnosis names a specific change record, a specific precedent incident and a specific runbook section, each with a retrieval score — and the on-call engineer can open all three before acting. If answers come back without citations, or citations that do not support the claim, treat that exactly as you would in [Lab 04 — Ground answers in your own guidance](#): the retrieval threshold or the corpus needs work, and the model's fluency is not a substitute.

The representative MTTR decomposition:

	Detect	Diagnose	Fix	Verify	Total
Four quarters before	4m	2h 41m	22m	9m	3h 16m
With the notebook	4m	24m	22m	9m	59m



Generated incident summaries are draft artifacts. They become operations knowledge for the next event only after the same curation and review that any other knowledge record gets — do not let fluency become authority.

15.6. Troubleshooting

logcli returns nothing

Check the tenant header and that the LokiStack gateway route is reachable from the workbench.
`oc -n openshift-logging get lokistack` first.

The notebook answers without citations

Same failure mode as [Lab 04 — Ground answers in your own guidance](#): the retrieval score threshold is too low or the corpus is missing the relevant material. Ingest the runbooks before you trust the answers.

Ingest is slow or huge

Summarize log windows before embedding rather than embedding raw lines — the notebook needs the shape of the incident, not every log line ever written.

15.7. Discussion questions

1. Where do your runbooks, postmortems and change records live today, and who would own keeping the notebook's corpus current?
2. What approval gate do you want between a cited recommendation and an automated remediation — and does [Lab 05 — Automate the toil around the model](#)'s rulebook already express it?
3. Which incident from the last year would this have shortened, and by how much?

Chapter 16. Lab 08 — One loop, end to end

Product	Red Hat OpenShift AI, Ansible Lightspeed, Event-Driven Ansible
Rung	data → information → knowledge → judgement
Time	About 8 minutes
Script	labs/lab08-grand-demo/run.sh
Outcome	One platform carrying logs, scores, drafts, review, automation and answers, with the loop as the knowledge base

16.1. Objective

Mainframe and server logs come in, a predictive model scores which systems will fall over next, Ansible Lightspeed drafts the rulebook and playbook, a person reviews and merges them, Event-Driven Ansible runs them, and the assistant answers "how is server8-audits?" by probing the estate. One platform carries every step, and the loop is also the knowledge base.

On the ladder this lab climbs every rung in order. The log lines are data; the outage-risk score is information about one system; the drafts, the review thread and the job outcome become knowledge the moment they are merged and indexed; and the merge itself is judgement, a determination a named reviewer signs. Automation executes that judgement. It never makes it.

This is the lab that ties the other seven together: the tabular model from [Lab 02 — Teach it your notice taxonomy](#), model serving from [Lab 03 — Survive filing season](#), retrieval from [Lab 04 — Ground answers in your own guidance](#), the rulebook and job history from [Lab 05 — Automate the toil around the model](#), and the operations corpus from [Lab 07 — Ask your own logs](#), on the one platform [Lab 01 — Serve a model inside your boundary](#) and [Lab 06 — Prove it to your ISSO](#) keep inside the boundary.

16.2. What you use

- Two model pods side by side on **OpenShift AI**: a tabular outage-risk scorer (sklearn on KServe) and Granite on vLLM, one namespace, one GPU node
- **Vector** and **Loki** ingesting z/OS SYSLOG, RHEL journald and OpenShift container logs into one queryable store
- **Ansible Lightspeed** (on-prem, its model served by OpenShift AI) drafting the playbook and rulebook, then a pull request for the human review
- **Event-Driven Ansible** running the promoted rulebook, and the ops assistant probing hosts through approved job templates

16.3. What to notice

- The scorer produces a **number**; the drafter produces **YAML**; neither runs anything until a person merges the pull request

- The merge is the promotion: repo, AAP project and knowledge base update in one step
- Every assistant answer carries citations and a job id: evidence, not fluency
- Nothing left the cluster: mainframe logs, scores, drafts and answers all stayed inside the boundary

16.4. Prerequisites

For live mode: Red Hat OpenShift AI with a KServe sklearn runtime for the outage-risk model and vLLM for Granite; OpenShift Logging with Vector receiving z/OS SYSLOG beside journald and container logs into Loki; Red Hat Ansible Automation Platform 2.5 with Ansible Lightspeed on-prem, its model served through OpenShift AI model serving; an Event-Driven Ansible controller with the named job templates; an ITSM webhook for the ticket; and `oc`, `logcli`, `python3`, `ansible-rulebook`, `ansible-lint`, `git`, `gh` and `awx` on the path.

As with [Lab 07 — Ask your own logs](#), the helpers the script calls (`ingest_estate.py`, `score.py`, `lightspeed_draft.py` and `ask.py`) are representative wrappers and are not shipped in the lab directory. What is shipped beside `run.sh` is the Lightspeed prompt (`prompts/memory-pressure.txt`) and the reviewed, post-merge rulebook and playbook (`rulebooks/memory_pressure.yml`, `playbooks/relieve_memory_pressure.yml`).

```
cd labs/lab08-grand-demo
LAB_PACE=step ./run.sh
```

16.5. Walkthrough

Step	Command	What to look for
1	<code>oc get pods -n tax-loop -o wide</code>	Two kinds of AI in one namespace: the sklearn scorer and the Granite drafter both on <code>ocp-gpu-02</code> , beside Vector, Loki, the EDA worker and the assistant. Same registry, same identity.
2	<code>logcli query '{source=~"zos-syslog journald openshift"} ~ "OOM S878 S80A OOMKilled"'</code>	Three estates, one query: an S878 abend on MVSA, two OOM kills on <code>server8-audits</code> , an OOMKilled container in <code>tax-inference</code> , and the survivor on <code>server1-taxreturns</code> whose cgroup limit held.
3	<code>python ingest_estate.py --sources zos-syslog,journald,openshift --window 30d</code>	2,209,060 lines become 10,080 rows of hourly features per system. Same Loki the Lab 07 — Ask your own logs notebook reads; nothing new was installed and no line left the cluster.

Step	Command	What to look for
4	<code>python score.py --model outage-risk --horizon 72h --top 5</code>	A number per system: <code>server8-audits</code> at 0.91 and rising. Nothing has been decided; a person decides what 0.91 means tonight. The model is the Lab 02 — Teach it your notice taxonomy tabular pattern, AUC 0.93 on 180 days of history.
5	<code>python lightspeed_draft.py --prompt prompts/memory-pressure.txt</code>	A draft playbook and rulebook with citations to RB-207, CR-9120 and INC-2311. Nobody has run it. Lightspeed's model is served by the same OpenShift AI as the scorer.
6	<code>ansible-lint drafts/ && git commit ... && gh pr create -r sre-lead</code>	The human gate. The reviewer asks for two edits (threshold 0.8 → 0.85, a <code>maintenance_window</code> guard), approves and merges. One merge syncs the AAP project, restarts the activation and indexes the review thread as RB-214.
7	<code>ansible-rulebook --rulebook rulebooks/memory_pressure.yml</code>	The loop closes: <code>OutageRiskHigh</code> at 03:12:51, job 40512 raises Xmx 6g to 7g inside the 9g cgroup, restarts the service, probes 200 OK, opens and closes INC-2340. Forty-two seconds, no page. Same job history as Lab 05 — Automate the toil around the model .
8	<code>python ask.py "How is server8-audits? ..."</code>	Three probes through approved job templates, three cited answers. The assistant changes nothing except through a job template a person approved.

16.6. What good looks like

The score is a number with drivers, not a verdict:

SYSTEM	72H RISK	DRIVERS
TREND		
server8-audits	0.91	heap headroom -38% wk/wk · GC p95 2.4s · 2 OOM kills
^ rising		
MVSA / IMFBATCH	0.74	S878 x3 in 7d · region 512M · batch overlap 02:00-03:00
^ rising		
server3-notices	0.41	swap in use 18% · restarts 1
= flat		
server1-taxreturns	0.22	cgroup limit held · 1 oom-killer survived
v falling		
server5-transcripts	0.09	-
= flat		
Model: outage-risk v14 trained 2026-09-01 on 180d of estate features AUC 0.93 (tabular; the Lab 02 pattern)		

The review is visible, and the merge does three jobs at once:

```
https://git.agency.gov/ops/loop-ops/pull/482
  review sre-lead requested changes: threshold 0.8 -> 0.85; add maintenance_window
guard (+11 min)
  review sre-lead approved after 2 edits
(14:07)
  merged main <- eda/memory-pressure #482 by j.rivera
awx project update loop-ops -> synced rev 3f1c2a9 (14:08)
awx rulebook activation restart memory-pressure -> running, rulebook
memory_pressure.yml
km_publish: PR #482 + review thread indexed as RB-214 in pgvector.ops_notebook (cites
CR-9120, INC-2311)
```

The remediation runs only the approved playbook, and records itself:

```
[03:12:51] EVENT OutageRiskHigh instance=server8-audits score=0.91
[03:12:51] RULE "Outage risk above threshold" matched (maintenance_window: true)
[03:12:52] ACTION run_job_template -> "Relieve memory pressure" host=server8-audits
[03:13:29] job 40512 | Xmx 6g -> 7g (cgroup 9g) | audits-case.service
restarted | probe 200 OK in 0.31s
[03:13:33] job 40512 | ServiceNow INC-2340 opened and closed | KM record RB-
214 updated (outcome: success)
Elapsed from event to verified: 42 seconds. No page. server8-audits re-scored: 0.91 ->
0.34
```

The assistant probes through job templates and cites what the loop wrote:

```
[tool] job_template.launch "Probe host" host=server8-audits -> job 40519 3.8s
server8-audits: online, restored 03:13 by job 40512 (Xmx 7g); no OOM since; risk 0.91
-> 0.34;
change PR #482 merged, INC-2340 closed. [job 40512 · PR #482 · INC-2340]
[tool] job_template.launch "Estate ping" hosts=all -> job 40520 6.2s
online 13/14: server1..server12 ok · MVSA reachable (z/OSMF) · MVSB in planned IPL
window until 06:00 [CR-9131]
```

If a score arrives without drivers, a draft arrives without citations, a job runs that nobody merged, or an answer arrives without a job id, that is the finding. Each of those is the same failure the earlier labs guard against, and the loop is only as trustworthy as its weakest gate.

16.7. Two kinds of mathematics, one platform

Two different mathematics share one platform. The outage-risk model is tabular prediction: hourly features per system (memory headroom, GC pauses,abend rate, batch overlap) fitted to 180 days of history, so its output is a probability, not an explanation. The drafts come from a code model predicting tokens, steered by your prompt and the runbooks retrieved as context, so its output is

plausible YAML that must be reviewed like any other change. Event-Driven Ansible contains no machine learning at all: the same event always runs the same approved job. The knowledge base grows because every promotion (the pull request, its review thread, the job outcome) is embedded into the same vector store the assistant searches, which is why the assistant can cite RB-214 an hour after it was written.



One platform runs the predictive model, the generative model, the automation and the knowledge base, so the loop that keeps systems up is also the documentation, and the assistant answers from what the loop wrote.

16.8. Troubleshooting

logcli returns only one of the three sources

The label is **source**. Check the Vector pipeline receiving z/OS SYSLOG and the journald forwarder on the RHEL hosts; `oc -n openshift-logging get clusterlogforwarder` shows what is wired.

The scorer answers but every risk is 0.0x

The feature window is too short or the model was trained on a different estate. Re-run the ingest with the 30-day window and confirm the model version in the registry matches **outage-risk v14**.

The draft has no citations

Lightspeed is answering from the model alone. Point it at the [Lab 07 — Ask your own logs](#) operations corpus so RB-207, CR-9120 and INC-2311 are retrievable, and treat an uncited draft the way [Lab 04 — Ground answers in your own guidance](#) treats an uncited answer.

The rule matches but no job launches

The job template name and organization must match Automation Controller exactly, and the activation must have restarted after the merge. Same troubleshooting as [Lab 05 — Automate the toil around the model](#).

The playbook refuses to restart the service

That is the **maintenance_window** guard from the review. Outside the window the second rule opens a change request instead; a person decides.

The assistant answers without a job id

It is answering from memory rather than probing. Every host answer should carry a **job_template.launch** line; if it does not, the tools are not wired and the answer is fluency, not evidence.

16.9. Discussion questions

1. Which of the eight steps does your agency already have, and which seam between them costs the most time today?
2. Who is the **sre-lead** in your organization, and what would it take for a Lightspeed draft to reach them as a pull request rather than a ticket?
3. Which events would you let close the loop automatically, which only inside a maintenance

window, and which should always stop at a person?

Chapter 17. Facilitator guide

For whoever is running the session. Attendees do not need this section.

17.1. Scope

This session is for the IRS audience. Do not forward the invite to other accounts. Ted Craig (Red Hat account executive for the IRS) asked that the focus stay on IRS for this one; other AEs replicate it for their own accounts with Brad and their partners, and it should be duplicated. If someone from outside the IRS turns up anyway, the answer is the same one Brad gives at the close: reach out, and a similar session can be given for their mission.

17.2. Before Thursday

Owners and deadlines from the September 15 review.

- Send the webinar join link to the internal team, Nathan Holland, Madhu Kuriti and Michael Massaro, before Thursday. Owner: Brad. This is the internal team, not registrants.
- Confirm whether the public GitHub short URL may be shown live or needs a blessing first. Owner: Brad. The question was asked at the review and nobody knew. Fallback if it is still open on the day: show the Carahsoft-hosted HTML on screen and say the repo link goes out in the follow-up.
- Day of, Brad and Jon Keam review the alternate single-focus graphics and swap one in wherever Jon flagged that one picture beats several. Slide 6 already carries the single predictive/generative overlap graphic (the six-row table survives in the reveal export as the take-home); slide 4 (the engine) is the next candidate. The five illustrated renders pop out with **6** on the slides that have one; drop the full-resolution files into `slides/assets/graphics/` and the pop-outs show them instead of the vector renditions.
- Confirm whether anyone from IRS cyber is registered. We do not know who the 52 are, so plan the ask at the close either way: is anyone on from cyber? If yes, say security is covered at the very end and offer to meet separately to go deeper. If no, ask whether anyone can refer us to someone in cyber who would want that conversation.

17.3. Timing a 60-minute virtual event

Minutes	Content	Mode
0–3	Welcome — Brad drives, Jon color-commentates. Walk the pyramid bottom to top and say the rule on its base: solve a problem someone has. Set the promise: every claim gets run in a shell.	Slide 1
3–11	Foundations first, so the room shares a vocabulary: terminology, why now, inside an LLM, data-to-judgement, predictive vs generative.	Slides 2–6

Minutes	Content	Mode
11–17	Where AI lands, the five outcomes, the reference architecture (open with the seams riff).	Slides 7–9
17–23	Why Red Hat AI (glass box, Models-as-a-Service) and define-once-run-anywhere.	Slides 10–11
23–28	The operations loop and logs-to-MTTR — industry metric first, then the personal story.	Slides 12–13
28–32	Adoption path (crawl, walk, run) and next steps — point at the contact strip.	Slides 14–15
32–34	Labs preview — the concluding takeaway. Switch to the Labs view.	Slide 16
34–42	Run Lab 01 live on screen. Stop on <code>ss -ltnp</code> . That single line answers the data question.	Lab 01
42–50	Run Lab 05 live on screen. The 62-second remediation is the moment people remember.	Lab 05
50–56	Skim Labs 02, 03, 04, 06, 07. Tell them the repo is theirs to take home.	Labs
50–58	Optional finale for an operational room: run Lab 08 live in place of the skim (slide 17 is its map).	Slide 17 · Lab 08
56–60	Questions. The security appendix stays on standby — pull it up when an ISSO engages.	Slide 18



If you only have 30 minutes, cut to slides 1, 2, 8 and run Lab 01 plus the drift section of Lab 06. Keep the appendix (slide 18) loaded for security questions.

Lab 08 is the finale for an operational room. Eight minutes, the whole loop on slide 17: logs in, risk scored, Lightspeed drafts, a person merges, EDA runs, and the assistant answers with job evidence. Run it in place of the 50 to 56 skim; Lab 01 and Lab 05 stay the live ones. Two takeaways to say out loud: the platform is also the KM system and the documentation, because every promoted change is a cited record; and the assistant launches approved jobs and reports, it changes nothing itself. Then the question to ask the room: what is the last runbook you wrote that the on-call actually found?

Security posture (slide 18) stays the appendix. Brad offered at the September 15 review to move it ahead of the labs and next steps; Ted Craig (Red Hat account executive for the IRS) decided it stays where it is, because cyber will be one or two of the 52 or so at most, not the majority of the room by any stretch. Pull it up only when a cyber person engages, and offer the cyber team a separate, deeper session rather than reshaping this one.

17.4. Delivery

Brad drives; Jon is the color commentator — interrupt, add commentary, call out misspeaks live, and take over if the driver flounders. The number one goal is to entertain: a tuned-out audience learns nothing. Keep it light, and if humour lands on anyone, self-deprecate first.

Five planted questions are rehearsed; the console marks each one PLANTED. On the data-to-judgement slide (5) Jon asks "so is AI taking our jobs?" — the answer is that it is not, because it has no judgement. On the architecture slide (9) Jon plays the foil with "what's a seam?" (the pants analogy) and Brad asks Jon "what is fine-tuning to you?" — the studying-for-the-test answer. On the why-one-platform slide (11) Jon challenges lock-in ("I thought we already bought the software" — the phone-plan analogy), and on the adoption-path slide (14) Jon asks what a predicament is versus a problem. Plan deliberate pauses to engage the room.

The theme that holds the hour together is the ladder: data → information → knowledge → judgement. Every slide shows the rungs it serves in the top-right spine, and every technology, outcome and lab carries a rung chip in the same four colours (blue, amber, red, green). Use it as the connective tissue: when a product name lands on screen, say which rung it climbs and who climbs the next one. Judgement is always a person; automation executes it.

The full talk track, including the seams, lock-in and MTTR riffs, lives in [slides/presenter-notes.html](#) — open it beside the deck and advance both together. Each console step lists the slide's rungs and an "On the ladder" beat mapping every item on the slide to its rung.

17.5. Which labs to run live

Lab 01

Always. Step 4 — the endpoint bound to **127.0.0.1** — is the single most persuasive screen in the whole deck for a tax agency audience.

Lab 05

Almost always. Watching an alert resolve itself in 62 seconds lands harder than any slide about automation.

Lab 06

If security staff are in the room, run steps 3 to 5. The deliberate **NON-COMPLIANT** result and the drift misreading are both good discussion.

Lab 04

Run step 4 — the refusal — if anyone raises hallucination. It is a 30-second answer to a question that otherwise consumes ten minutes.

Lab 07

If operations or SRE leaders are in the room. The cited diagnosis in step 3 and the MTTR table in step 5 are the two screens that sell the ops-notebook story.

Lab 08

When the room wants the whole loop rather than one link of it. Steps 4, 6 and 7 (the score, the merged review, the 42-second remediation) carry the argument; step 8, the assistant probing the estate, is the closer if time allows.

Labs 02 and 03

Usually better as a walkthrough than a live run. Both have long-running steps that do not demo well in real time.

17.6. Running the labs on a projector

```
# Larger pauses, no colour issues on unfamiliar terminals
LAB_SPEED=0.08 LAB_PACE=step ./labs/lab01-rhel-ai/run.sh

# If the room's screen share mangles ANSI colour
NO_COLOR=1 LAB_PACE=step ./labs/lab01-rhel-ai/run.sh
```

Increase your terminal font size before the session, not during it. Eighty columns is enough for every lab — none of the output wraps at that width.

If you would rather not type at all, open the interactive deck ([slides/deck.html](#)) and switch to its **Demo** view — it plays every lab hands-free with play/pause, per-lab jumps and a speed control, and each lab terminal also has its own **Demo** button. Opening the page with `#demo` on the URL starts the Demo view playing on a continuous loop, which doubles as a holding screen before the session begins. For screen sharing, the **Present** button (or **F**) fullscreens the 16:9 slide stage.

17.7. The deck's switches

The interactive deck carries an **EN/ES** language tab in the sidebar footer — Spanish audiences get the slides, lab briefings, theory panels and glossary translated, while terminal commands and output stay English, as real tools do. Next to it sit a light/dark **theme** toggle and the **Notes** button. Every lab carries a theory panel ("what just happened"), and a glossary closes the Labs view — when someone asks where an answer came from, the foundations slides plus the Lab 01 theory panel are the prepared response.

As each command runs, a **step card** pops up under the terminal (beside it in the Demo view) with three facets: what is happening, why it matters to this talk, and the engineering seam a single Red Hat platform removes — the hand-built box nobody documented, the glue script between alert and fix, the scanner report nobody can diff. A wrap card closes each lab with what was just proved. The cards are the talk track: read the middle facet aloud when the room is mission-minded, the right-hand one when it is procurement-minded. They follow the EN/ES tab like everything else.

Every slide has two faces. **Tell** is the word-heavy version; **Show** keeps the central graphic and drops the paragraphs, so you talk to the picture. The toggle sits in the bar under the stage and on **S**; it is remembered per browser. Present in Show, switch to Tell when someone asks for the detail, and leave the deck in Tell for the take-home copy.

Two keys are deliberately not printed on the deck. **N** shows the speaker notes on the shared screen: do not press it while sharing. **P** opens the presenter console ([presenter-notes.html](#)) in its own window and keeps it on the current slide or lab as you move; put that window on the presenter's display only. The console carries the read-aloud script, the discussion notes, the Jon cues, the graphic talking points and the lab step cards side by side.

Two more switches arrived after the September 15 review. The ladder now sits in the corner of every slide as a small **stacked pyramid** (judgement at the apex, data at the base, the rainbow's yellow and blue as thin bands between); the rungs a slide serves are lit, the rest dimmed, so the

room can track where each slide lives on the climb without a legend. The title slide carries the same pyramid at full size, each layer with the question it answers (what happened; what it means for this case; what we know that bears on it; what we will do and who signs), a rail that says data informs every layer, the end state at the apex (knowing enough to decide) and the rule along its base: solve a problem someone has; don't invent a solution to one no one has. Walk it bottom to top before anything else. And the slides that have an illustrated graphic (title, data to judgement, where AI lands, five outcomes, architecture, why one platform, labs preview, next steps) carry a small **graphic** pill; press **G**, the pill, or the Graphic button under the stage to pop the picture out over the deck with three or four discussion points beside it, then **Esc** or **G** again to close. The Labs view opens the "Choose your path" map the same way. The pop-out shows the illustrated render when the file is present in `slides/assets/graphics/` and a vector rendition of the same content otherwise.

17.8. Questions you will get, and short answers

"Can this run air-gapped?"

Yes. Mirror images and models into the enclave with `oc-mirror`. The platform is built for disconnected operation; it is a supported configuration, not a workaround.

"What does it cost in GPUs?"

An 8B model serving a pilot workload fits on one accelerator. Filing-season concurrency is the number that sets the ceiling, and almost nobody knows theirs. That measurement is a good first engagement.

"How do we stop it hallucinating?"

You constrain it with retrieval and you let it refuse. Show Lab 04, step 4.

"Is our data used to train anything?"

No. Inference runs on your hardware. Retrieval queries your corpus without learning it. Tuning happens only on data you deliberately place in the taxonomy.

"Can we use a different model?"

Yes. The serving layer is model-agnostic — KServe and vLLM will serve what you bring. The support relationship differs, which is worth being precise about.

"What about FedRAMP?"

Ask which boundary they are operating in before answering. The honest response is that the platform runs inside accredited environments and the accreditation belongs to the environment, not to the software.

"Is there an in-person version of this?"

Tease it, do not date it. A larger on-site follow-on at New Carrollton in late October or early November is being planned on Ted Craig's side, and nothing about it goes on a slide yet. Say "watch for an in-person session this fall" and leave it there.

17.9. Facilitator's honesty checklist

Say these out loud during the session. They buy more credibility than any demo.

- The sample notice content is illustrative, written for the exercise. No agency reviewed it.
- The numbers in simulated output are representative of a working environment, not measurements from a named deployment.
- Exam selection is deliberately not on the list of good first candidates. Determinations affecting a taxpayer need governance long before automation.
- Every workflow in the deck keeps a named human on the signature. The model drafts; it never sends.

17.10. Pilot requests

The first response to any proof-of-concept interest is a scheduled show-and-tell: Brad and Jon Keam (or whoever Brad picks) walk the eight labs with the interested team to find where in the stack they want to prove value right now, and to start the community of practice around it. Only if the POC gets serious or ratchets up do we bring in more resources (Dan and company).

Route the request through the account executive. For the IRS that is Ted Craig (Carahsoft), who is on the call. Anyone from outside the IRS, or anyone who does not know who their AE is, emails Brad and gets connected to their equivalent of Ted, and the session is replicated with their own account team.

17.11. After the session

1. Share the site link — <https://nommsweymx.github.io/redhat-ai-tax-labs/> — and the repository. The labs are the same commands attendees will run on day one.
2. Tell current Red Hat customers to ask for their Solution Architect; anyone who does not know who that is can email Brad (bscalio@redhat.com) or Jon (jkeam@redhat.com), open a ticket in the Customer Portal and ask support, or ask through their management line.
3. Offer to help seed a community of practice across platform, security and mission teams, built around one workflow.
4. Offer the architecture workshop — half a day with platform and security staff to size accelerators, place the first workload and name the data classes in scope.
5. Bring Four Inc. and Carahsoft into the acquisition conversation early rather than at the end.

17.12. Follow-up ownership

56 registered: 52 via Ted Craig and Caleb's Carahsoft outreach, 4 via Four Inc. Most are contacts we have not been talking to before, which is the point. Carahsoft sends the full contact list to Ted Craig (Red Hat account executive for the IRS) at the end; Caleb (Red Hat inside sales) runs the follow-up campaign with the recording, the deck and the labs link, plus any document links worth attaching. Brad's action: hand Caleb the deck and the links before the campaign goes out.