pydantic_ai.models.vertexai

Custom interface to the `*-aiplatform.googleapis.com` API for Gemini models.

This model uses `GeminiAgentModel` with just the URL and auth method changed from `GeminiModel`, it relies on the VertexAI `generateContent` and `streamGenerateContent` function endpoints having the same schemas as the equivalent Gemini endpoints.

Setup

For details on how to set up authentication with this model as well as a comparison with the `generativelanguage.googleapis.com` API used by `GeminiModel`, see model configuration for Gemini via VertexAI.

Example Usage

With the default google project already configured in your environment using "application default credentials":

vertex_example_env.py

```python
from pydantic_ai import Agent
from pydantic_ai.models.vertexai import VertexAIModel

model = VertexAIModel('gemini-1.5-flash')
agent = Agent(model)
result = agent.run_sync('Tell me a joke.')
print(result.data)
#> Did you hear about the toothpaste scandal? They called it Colgate.
```

Or using a service account JSON file:

vertex_example_service_account.py

```python
from pydantic_ai import Agent
from pydantic_ai.models.vertexai import VertexAIModel

model = VertexAIModel(
    'gemini-1.5-flash',
    service_account_file='path/to/service-account.json',
)
agent = Agent(model)
result = agent.run_sync('Tell me a joke.')
print(result.data)
#> Did you hear about the toothpaste scandal? They called it Colgate.
```

VERTEX_AI_URL_TEMPLATE `module-attribute`

```python
VERTEX_AI_URL_TEMPLATE = "https://{region}-aiplatform.googleapis.com/v1/projects/{project_id}/locations/{region}/publishers/{model_publisher}/models/{model}:"
```

URL template for Vertex AI.

See `generateContent` docs and `streamGenerateContent` docs for more information.

The template is used thus:

- `region` is substituted with the `region` argument, see available regions
- `model_publisher` is substituted with the `model_publisher` argument
- `model` is substituted with the `model_name` argument
- `project_id` is substituted with the `project_id` from auth/credentials
- `function` (`generateContent` or `streamGenerateContent`) is added to the end of the URL

VertexAIModel `dataclass`

Bases: `Model`

A model that uses Gemini via the `*-aiplatform.googleapis.com` VertexAI API.

Source code in `pydantic_ai_slim/pydantic_ai/models/vertexai.py`

```python
@dataclass(init=False)
class VertexAIModel(Model):
    """A model that uses Gemini via the `*-aiplatform.googleapis.com` VertexAI API."""

    model_name: GeminiModelName
    service_account_file: Path | str | None
    project_id: str | None
    region: VertexAiRegion
    model_publisher: Literal['google']
    http_client: AsyncHTTPClient
    url_template: str

    auth: BearerTokenAuth | None
    url: str | None

    # TODO __init__ can be removed once we drop 3.9 and we can set kw_only correctly on the dataclass
    def __init__(
        self,
        model_name: GeminiModelName,
        *,
        service_account_file: Path | str | None = None,
        project_id: str | None = None,
        region: VertexAiRegion = 'us-central1',
        model_publisher: Literal['google'] = 'google',
        http_client: AsyncHTTPClient | None = None,
        url_template: str = VERTEX_AI_URL_TEMPLATE,
    ):
        """Initialize a Vertex AI Gemini model.

        Args:
            model_name: The name of the model to use. I couldn't find a list of supported Google models, in VertexAI
                so for now this uses the same models as the [Gemini model][pydantic_ai.models.gemini.GeminiModel].
            service_account_file: Path to a service account file.
                If not provided, the default environment credentials will be used.
            project_id: The project ID to use, if not provided it will be taken from the credentials.
            region: The region to make requests to.
            model_publisher: The model publisher to use, I couldn't find a good list of available publishers,
                and from trial and error it seems non-google models don't work with the `generateContent` and
                `streamGenerateContent` functions, hence only `google` is currently supported.
                Please create an issue or PR if you know how to use other publishers.
            http_client: An existing `httpx.AsyncClient` to use for making HTTP requests.
            url_template: URL template for Vertex AI, see
                [`VERTEX_AI_URL_TEMPLATE` docs][pydantic_ai.models.vertexai.VERTEX_AI_URL_TEMPLATE]
                for more information.
        """
        self.model_name = model_name
        self.service_account_file = service_account_file
        self.project_id = project_id
        self.region = region
        self.model_publisher = model_publisher
        self.http_client = http_client or cached_async_http_client()
        self.url_template = url_template

        self.auth = None
        self.url = None

    async def agent_model(
        self,
        *,
        function_tools: list[ToolDefinition],
        allow_text_result: bool,
        result_tools: list[ToolDefinition],
    ) -> GeminiAgentModel:
        url, auth = await self._ainit()
        return GeminiAgentModel(
            http_client=self.http_client,
            model_name=self.model_name,
            auth=auth,
            url=url,
            function_tools=function_tools,
            allow_text_result=allow_text_result,
            result_tools=result_tools,
        )

    async def _ainit(self) -> tuple[str, BearerTokenAuth]:
        if self.url is not None and self.auth is not None:
            return self.url, self.auth

        if self.service_account_file is not None:
            creds: BaseCredentials | ServiceAccountCredentials = _creds_from_file(self.service_account_file)
            assert creds.project_id is None or isinstance(creds.project_id, str)
            creds_project_id: str | None = creds.project_id
            creds_source = 'service account file'
        else:
            creds, creds_project_id = await _async_google_auth()
            creds_source = '`google.auth.default()`'

        if self.project_id is None:
            if creds_project_id is None:
                raise UserError(f'No project_id provided and none found in {creds_source}')
            project_id = creds_project_id
        else:
            if creds_project_id is not None and self.project_id != creds_project_id:
                raise UserError(
                    f'The project_id you provided does not match the one from {creds_source}: '
                    f'{self.project_id!r} != {creds_project_id!r}'
                )
            project_id = self.project_id

        self.url = url = self.url_template.format(
            region=self.region,
            project_id=project_id,
            model_publisher=self.model_publisher,
            model=self.model_name,
        )
        self.auth = auth = BearerTokenAuth(creds)
        return url, auth

    def name(self) -> str:
        return f'vertexai:{self.model_name}'
```

__init__

```python
__init__(
    model_name: GeminiModelName,
    *,
    service_account_file: Path | str | None = None,
    project_id: str | None = None,
    region: VertexAiRegion = "us-central1",
    model_publisher: Literal["google"] = "google",
    http_client: AsyncClient | None = None,
    url_template: str = VERTEX_AI_URL_TEMPLATE
)
```

Initialize a Vertex AI Gemini model.

Parameters:

Name	Type	Description	Default
`model_name`	`GeminiModelName`	The name of the model to use. I couldn't find a list of supported Google models, in VertexAI so for now this uses the same models as the Gemini model.	*required*
`service_account_file`	`Path | str | None`	Path to a service account file. If not provided, the default environment credentials will be used.	`None`
`project_id`	`str | None`	The project ID to use, if not provided it will be taken from the credentials.	`None`
`region`	`VertexAiRegion`	The region to make requests to.	`'us-central1'`
`model_publisher`	`Literal['google']`	The model publisher to use, I couldn't find a good list of available publishers, and from trial and error it seems non-google models don't work with the `generateContent` and `streamGenerateContent` functions, hence only `google` is currently supported. Please create an issue or PR if you know how to use other publishers.	`'google'`
`http_client`	`AsyncClient | None`	An existing `httpx.AsyncClient` to use for making HTTP requests.	`None`
`url_template`	`str`	URL template for Vertex AI, see `VERTEX_AI_URL_TEMPLATE` docs for more information.	`VERTEX_AI_URL_TEMPLATE`

Source code in `pydantic_ai_slim/pydantic_ai/models/vertexai.py`

```python
def __init__(
    self,
    model_name: GeminiModelName,
    *,
    service_account_file: Path | str | None = None,
    project_id: str | None = None,
    region: VertexAiRegion = 'us-central1',
    model_publisher: Literal['google'] = 'google',
    http_client: AsyncHTTPClient | None = None,
    url_template: str = VERTEX_AI_URL_TEMPLATE,
):
    """Initialize a Vertex AI Gemini model.

    Args:
        model_name: The name of the model to use. I couldn't find a list of supported Google models, in VertexAI
            so for now this uses the same models as the [Gemini model][pydantic_ai.models.gemini.GeminiModel].
        service_account_file: Path to a service account file.
            If not provided, the default environment credentials will be used.
        project_id: The project ID to use, if not provided it will be taken from the credentials.
        region: The region to make requests to.
        model_publisher: The model publisher to use, I couldn't find a good list of available publishers,
            and from trial and error it seems non-google models don't work with the `generateContent` and
            `streamGenerateContent` functions, hence only `google` is currently supported.
            Please create an issue or PR if you know how to use other publishers.
        http_client: An existing `httpx.AsyncClient` to use for making HTTP requests.
        url_template: URL template for Vertex AI, see
            [`VERTEX_AI_URL_TEMPLATE` docs][pydantic_ai.models.vertexai.VERTEX_AI_URL_TEMPLATE]
            for more information.
    """
    self.model_name = model_name
    self.service_account_file = service_account_file
    self.project_id = project_id
    self.region = region
    self.model_publisher = model_publisher
    self.http_client = http_client or cached_async_http_client()
    self.url_template = url_template

    self.auth = None
    self.url = None
```

BearerTokenAuth `dataclass`

Authentication using a bearer token generated by google-auth.

Source code in `pydantic_ai_slim/pydantic_ai/models/vertexai.py`

```python
@dataclass
class BearerTokenAuth:
    """Authentication using a bearer token generated by google-auth."""

    credentials: BaseCredentials | ServiceAccountCredentials
    token_created: datetime | None = field(default=None, init=False)

    async def headers(self) -> dict[str, str]:
        if self.credentials.token is None or self._token_expired():
            await run_in_executor(self._refresh_token)
            self.token_created = datetime.now()
        return {'Authorization': f'Bearer {self.credentials.token}'}

    def _token_expired(self) -> bool:
        if self.token_created is None:
            return True
        else:
            return (datetime.now() - self.token_created) > MAX_TOKEN_AGE

    def _refresh_token(self) -> str:
        self.credentials.refresh(Request())
        assert isinstance(self.credentials.token, str), f'Expected token to be a string, got {self.credentials.token}'
        return self.credentials.token
```

VertexAiRegion `module-attribute`

```python
VertexAiRegion = Literal[
    "us-central1",
    "us-east1",
    "us-east4",
    "us-south1",
    "us-west1",
    "us-west2",
    "us-west3",
    "us-west4",
```

```
    "us-east5",
    "europe-central2",
    "europe-north1",
    "europe-southwest1",
    "europe-west1",
    "europe-west2",
    "europe-west3",
    "europe-west4",
    "europe-west6",
    "europe-west8",
    "europe-west9",
    "europe-west12",
    "africa-south1",
    "asia-east1",
    "asia-east2",
    "asia-northeast1",
    "asia-northeast2",
    "asia-northeast3",
    "asia-south1",
    "asia-southeast1",
    "asia-southeast2",
    "australia-southeast1",
    "australia-southeast2",
    "me-central1",
    "me-central2",
    "me-west1",
    "northamerica-northeast1",
    "northamerica-northeast2",
    "southamerica-east1",
    "southamerica-west1",
]
```

Regions available for Vertex AI.

More details here.